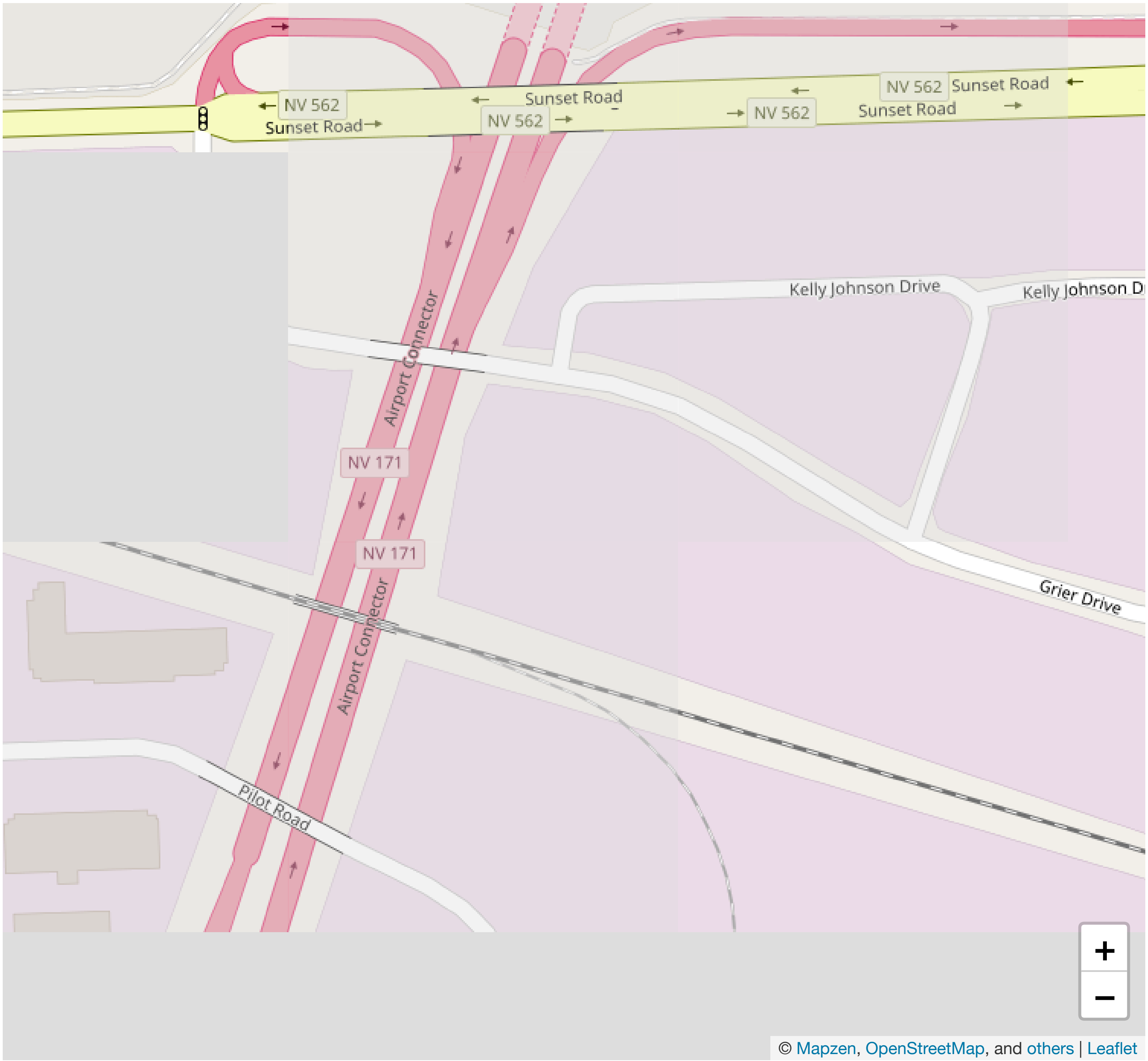
Meet Mapzen at CES

demo (/tag/demo)

Are you at CES this week? Come on by and say hi! We'll have demos running and will be ready to talk cartography, tools, and data for all your mapping needs.

We're in **Eureka Park (http://www.ces.tech/Show-Floor/Marketplaces/Eureka-Park)** at the Sands, **Hall G - 50231 (http://ces17.mapyourshow.com/7_0/floorplan/?hallID=M&selectedBooth=booth%7E50231)**, in front the Samsung Next booth.

(If you're at the airport, you can sit back and use TRON2.0 to **travel by map (https://bl.ocks.org/burritojustice/raw/dcf86abf2b5fe6c82b884f922d83212b/%22)** to get to the Strip.)


NV 562
Sunset Road
Sunset Road
NV 562
NV 562
NV 562
Sunset Road
Sunset Road
Kelly Johnson Drive
Kelly Johnson Dr
Airport Connector
NV 171
NV 171
Airport Connector
Grier Drive
Pilot Road
+
−

© Mapzen, OpenStreetMap, and others | Leaflet

· 04 January 2017 ·

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

Mike Cunningham

I'm Street View famous in two cities and I do product @mapzen.

© 2017 Mapzen

Lots of Dots

mapzen-js (/tag/mapzen-js) demo (/tag/demo) tutorial (/tag/tutorial) tangram (/tag/tangram) make-your-own (/tag/make-your-own)

Hello! And a warm welcome back to another installment of **Make Your Own (https://mapzen.com/tag/make-your-own/)** [Filters].

I hope you had a chance to ~~drink~~ eat too much, ~~argue~~ visit with family and generally enjoy the holidays. Our office was definitely quiet. (Let's hear it for grown-up work hours! 👏)

Well, I am happy to be back and **very happy** to be making more maps with you.

Today we will finally be leaving our geology map behind and entering the world of **dots**. Lots and lots of dots, in fact. More than you may even want to know about dots.

Are you excited?

Me too!

Did you want a refresher before we get started? Go ahead and take a look back at some of our previous ***Make Your Own (https://mapzen.com/tag/make-your-own/)*** posts. It's not a prerequisite, but there are some useful nuggets in there:

- **One Minute Map (https://mapzen.com/blog/one-minute-map/)**
- **Map Sandwich (https://mapzen.com/blog/map-sandwich/)**
- **Filters & Functions (https://mapzen.com/blog/filters-and-functions/)**
- **Put A Label On It (https://mapzen.com/blog/tangram-labels/)**
- **Interactive Mapping with Tangram (https://mapzen.com/blog/tangram-interactivity/)**

Ok, **now** we're ready. Let's go!

Data

I want to show you how to add lots of dots to a map. Not so many dots that we'll need to resort to tiling or clustering, but still quite a lot of dots.

We're going to look at the tree-lined streets of San Francisco. Or, at least the trees of those tree-lined streets, anyway. (**Khrushchev would be pleased. (https://burritojustice.com/2009/07/31/know-your-trees/)**)

Our **street trees dataset (https://data.sfgov.org/City-Infrastructure/Street-Tree-List/tkzw-k3nq)** comes from the **SF OpenData (https://data.sfgov.org/)** clearinghouse, a great source for data from the City and County of San Francisco.

To get these trees into a Tangram-readable format, I downloaded the CSV, stripped out any unnecessary columns, and converted to GeoJSON using **csv2geojson (https://github.com/mapbox/csv2geojson)**. This is the same tool that powers **http://geojson.io/ (http://geojson.io/)**, which I highly recommend if you have a smallish CSV with latitude and longitude data. Not so much for the 116,000+ tree dataset we are trying to convert now. Not that I didn't try... 😳

Lastly, I **minified (https://www.npmjs.com/package/json-minify)** the converted GeoJSON to bring the size down a few more MB. The dataset is still large (19MB) and best viewed on your laptop/desktop. But by stripping out extra columns and minifying, we were able to bring the final dataset down to less than a quarter of the original GeoJSON size. Woo!

With our conversion done and our GeoJSON in hand, we are ready to make a map!

Points

We'll be working entirely within a Tangram scene file today, so let's use **Tangram Play (https://mapzen.com/tangram/play/#13.0375/37.7613/-122.4296)**. Open up this link in a new tab or window so you can follow along:

https://mapzen.com/tangram/play/#13.0375/37.7613/-122.4296 (https://mapzen.com/tangram/play/#13.0375/37.7613/-122.4296)

The only prerequisite for using **Play (https://mapzen.com/tangram/play/)** is that our data source is accessible via a public link. For this reason, I put the GeoJSON file up in an S3 bucket (**here (https://s3.amazonaws.com/mapzen-assets/images/dot-**

mapping/trees_min.geojson)), but you can throw your data on any cloud storage device that allows you to share links publicly.

~~

I should stop here for just a second to tell you about a problem we're having. * *sigh* *

We are experiencing intermittent CORS (https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) issues with our s3 resources (including our trees dataset (https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/trees_min.geojson)). If you experience any hiccups while working through this example, try refreshing your browser. This typically fixes the issue. If that doesn't work, try clearing your cache or dropping us a line (mailto:hello@mapzen.com) and we'll help you debug.

Ok. Back to it. (And sorry for any inconvenience!)

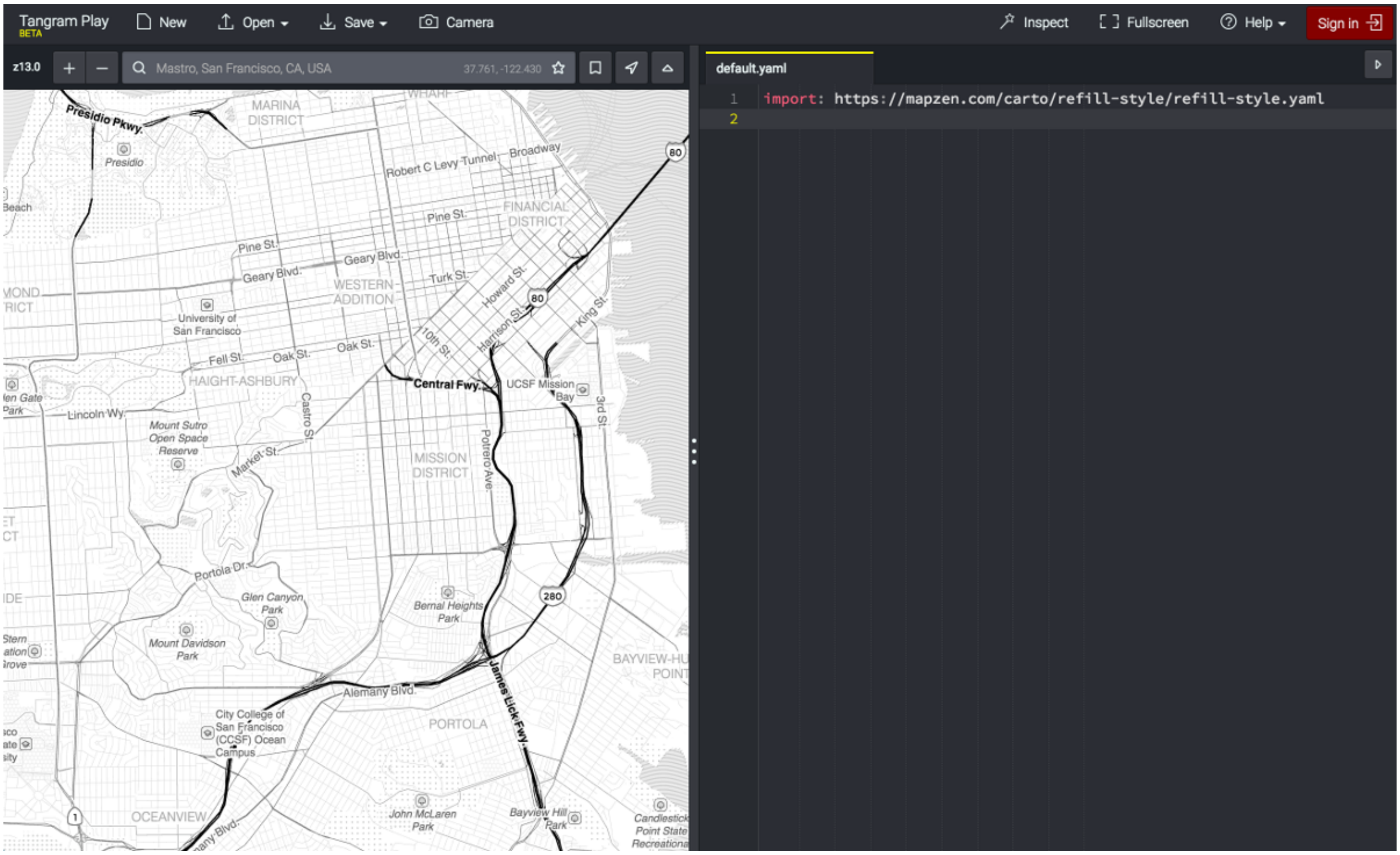
~~

If you'd rather build your map locally, feel free to keep your data local, too. Relative paths are totally acceptable in scene files.

Ok, let's take a look at Tangram Play. On the left side of the screen you should see a map of San Francisco. On the right, you'll see our `default.yaml` file. Go ahead and clear that file and replace with a single line:

```
import: https://mapzen.com/carto/refill-style/refill-style.yaml
```

The map on the left should automatically update, showing you the Mapzen **Refill (https://mapzen.com/documentation/cartography/styles/#refill)** basemap.

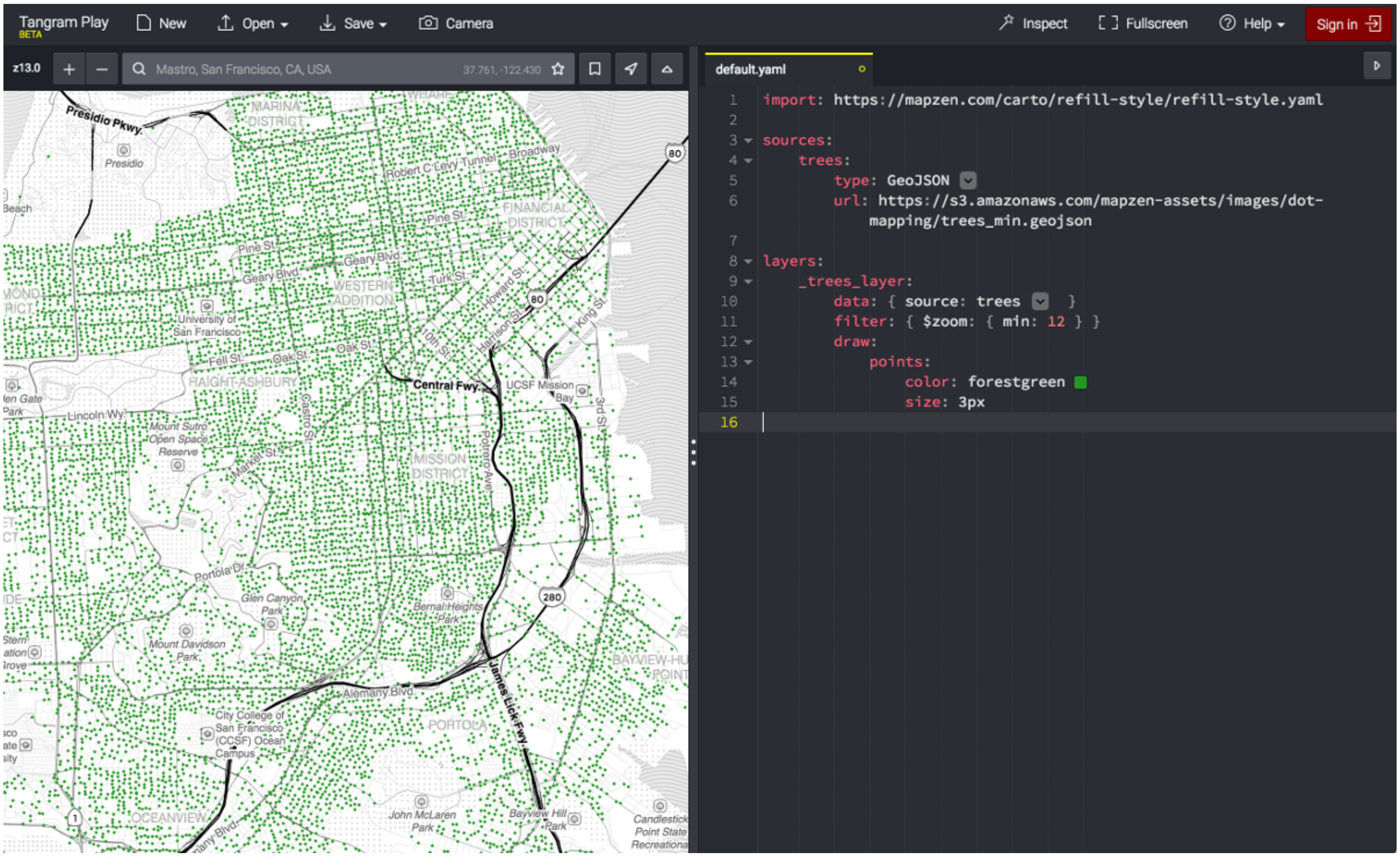

Tangram Play
BETA
New
Open
Save
Camera
Inspect
Fullscreen
Help
Sign in
z13.0
Mastro, San Francisco, CA, USA
37.761, -122.430
default.yaml
Presidio Pkwy.
Presidio
MARINA DISTRICT
Robert C Levy Tunnel
Broadway
Pine St
FINANCIAL DISTRICT
Geary Blvd
WESTERN ADDITION
Turk St
Howard St
Harrison St
King St
10th St
University of San Francisco
Fell St
Oak St
HAIGHT-ASHBURY
Central Fwy.
UCSF Mission Bay
3rd St
Lincoln Wy
Mount Sutro Open Space Reserve
Castro St
Market St
MISSION DISTRICT
Potrero Ave
Portola Dr
Glen Canyon Park
Mount Davidson Park
Bernal Heights Park
BAYVIEW-HU POINT
Alemany Blvd
James Lick Fwy.
City College of San Francisco (CCSF) Ocean Campus
PORTOLA
OCEANVIEW
John McLaren Park
Bayview Hill Park
Candlestick Point State Recreationa
80
280
1

```
import: https://mapzen.com/carto/refill-style/refill-style.yaml
```

The clean, black and white style of Refill will be a good starting place for our map.

Next, we'll import our trees data, define the trees layer, and give it a simple style:

```
sources:
    trees:
        type: GeoJSON
        url: https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/trees_min.geojson

layers:
    _trees_layer:
        data: { source: trees }
        filter: { $zoom: { min: 12 } }
        draw:
            points:
                color: forestgreen
                size: 3px
```


Tangram Play
BETA
New
Open
Save
Camera
Inspect
Fullscreen
Help
Sign in
z13.0
Mastro, San Francisco, CA, USA
37.761,-122.430
default.yaml
Presidio Pkwy.
MARINA DISTRICT
Robert C Levy Tunnel
Broadway
FINANCIAL DISTRICT
Geary Blvd
WESTERN ADDITION
University of San Francisco
HAIGHT-ASHBURY
Central Fwy
UCSF Mission Bay
MISSION DISTRICT
Mount Sutro Open Space Reserve
Glen Canyon Park
Bernal Heights Park
Mount Davidson Park
Alemany Blvd
PORTOLA
BAYVIEW-HUNTERS POINT
City College of San Francisco (CCSF) Ocean Campus
OCEANVIEW
John McLaren Park
Bayview Hill Park
James Lick Fwy
Candlestick Point State Recreationa

```
import: https://mapzen.com/carto/refill-style/refill-style.yaml

sources:
    trees:
        type: GeoJSON
        url: https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/trees_min.geojson

layers:
    _trees_layer:
        data: { source: trees }
        filter: { $zoom: { min: 12 } }
        draw:
            points:
                color: forestgreen
                size: 3px
```

Hey! Dots on a map!

Well, that was easy.

We are using one of Tangram's built-in draw styles called **points (https://mapzen.com/documentation/tangram/Styles-Overview/#points_1)**. The `points` draw style can be used to draw dots and **sprites (https://mapzen.com/documentation/tangram/draw/#sprite)**, or with a custom **shader (https://mapzen.com/documentation/tangram/shaders/)** or **texture (https://mapzen.com/documentation/tangram/textures/)**.

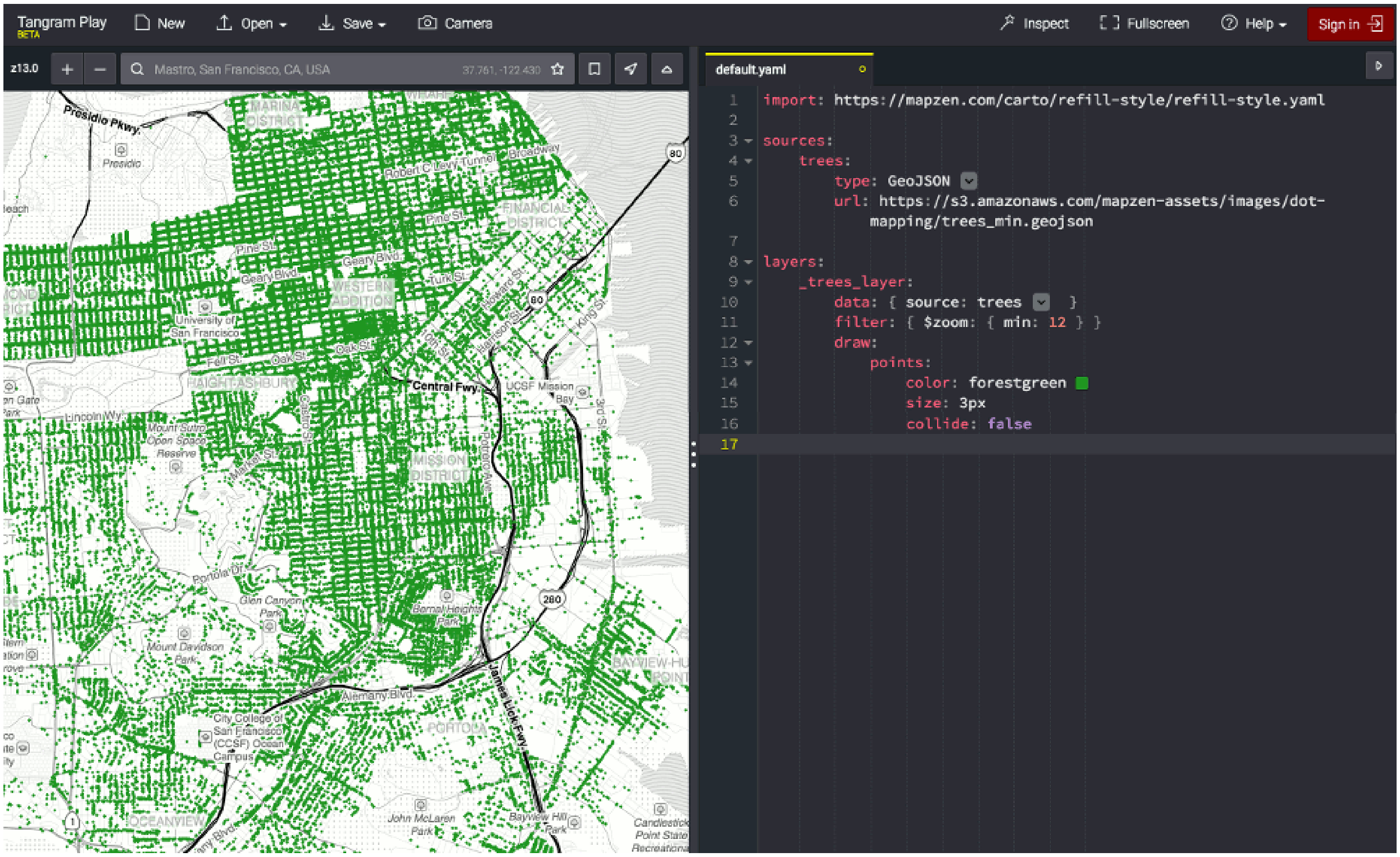
In our case, we're drawing a simple dot to represent almost every tree.

Wait. *Almost* every tree?

That's right. At many zoom levels, our points will *collide* with each other. By default, the `points` draw style determines that only one point from each of these *collisions* will be drawn. The winner is decided according to the **`priority` (https://mapzen.com/documentation/tangram/draw/#priority)** draw parameter.

Rather than limiting the number of points that are drawn, I'd rather show *all* trees, regardless of collisions. We can do that by setting the `collide` parameter to `false` :

```
        draw:
            points:
                color: forestgreen
                size: 3px
                collide: false
```


Tangram Play
BETA
New
Open
Save
Camera
Inspect
Fullscreen
Help
Sign in
z13.0
Mastro, San Francisco, CA, USA
37.761, -122.430
default.yaml
Presidio Pkwy
MARINA DISTRICT
FINANCIAL DISTRICT
WESTERN ADDITION
HAIGHT-ASHBURY
MISSION DISTRICT
Central Fwy
Lincoln Wy
Portola Dr
Glen Canyon Park
Bernal Heights Park
Alemany Blvd
James Lick Fwy
PORTOLA
OCEANVIEW
BAYVIEW-HUNTERS POINT
John McLaren Park
City College of San Francisco (CCSF) Ocean Campus

```
import: https://mapzen.com/carto/refill-style/refill-style.yaml

sources:
    trees:
        type: GeoJSON
        url: https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/trees_min.geojson

layers:
    _trees_layer:
        data: { source: trees }
        filter: { $zoom: { min: 12 } }
        draw:
            points:
                color: forestgreen
                size: 3px
                collide: false
```

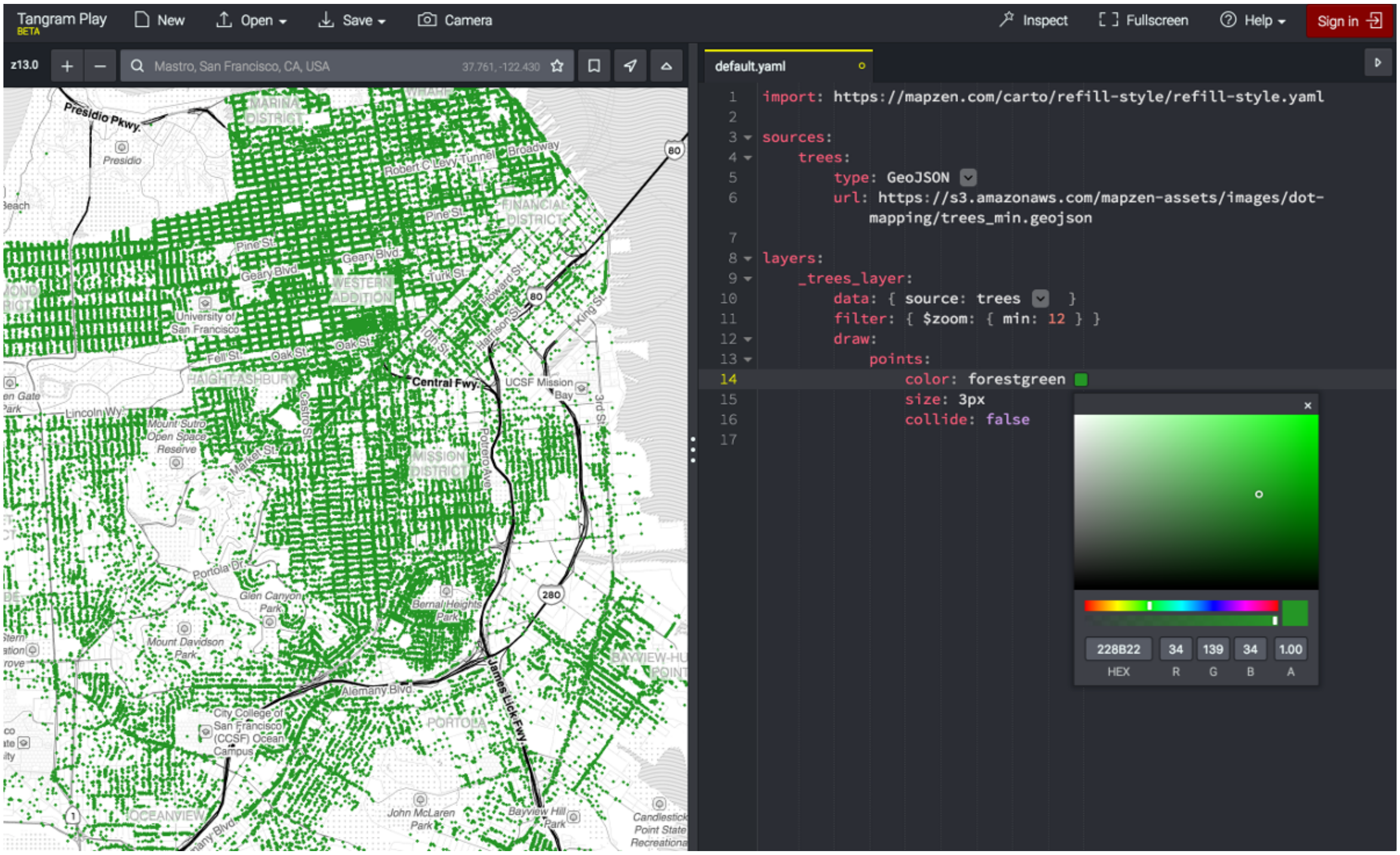
Woa. You can really see those 116,000 trees now, huh?

Let's add some transparency so we can see the trees that overlap (er, *collide*).

In Tangram, **color (https://mapzen.com/documentation/tangram/draw/#color)** can be represented in a wide variety of ways: color names (like `forestgreen`), hex, rgb/rgba, hsl/hsla, functions, **stops (https://mapzen.com/documentation/tangram/yaml/#stops)**, and one more we'll use now. It's rgba, but rather than values ranging between 0 and 255, the values range between 0 and 1. This format is used frequently in custom **shaders (https://mapzen.com/documentation/tangram/shaders/)** and can be found all throughout the **Tangram documentation (https://mapzen.com/documentation/tangram/)**.

The easiest way to convert other color formats to this RGBA[*number, number, number, number*] format is by using Tangram Play:

On the `color` line next to the word `forestgreen`, you should see a small green square. Click on the square to open the color picker:


Tangram Play BETA
New
Open
Save
Camera
Inspect
Fullscreen
Help
Sign in
z13.0
Mastro, San Francisco, CA, USA
37.761, -122.430
default.yaml

```
import: https://mapzen.com/carto/refill-style/refill-style.yaml

sources:
    trees:
        type: GeoJSON
        url: https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/trees_min.geojson

layers:
    _trees_layer:
        data: { source: trees   }
        filter: { $zoom: { min: 12 } }
        draw:
            points:
                color: forestgreen
                size: 3px
                collide: false
```

HEX	R	G	B	A
228B22	34	139	34	1.00

Select a color inside the color picker and close the box. You'll notice that the color value has changed to the RGBA[*number, number, number, number*] format. The first three numbers represent red, green, and blue. The fourth is an alpha value.

Change that `1.0` alpha value to `0.5` and watch as our points become semi-transparent.

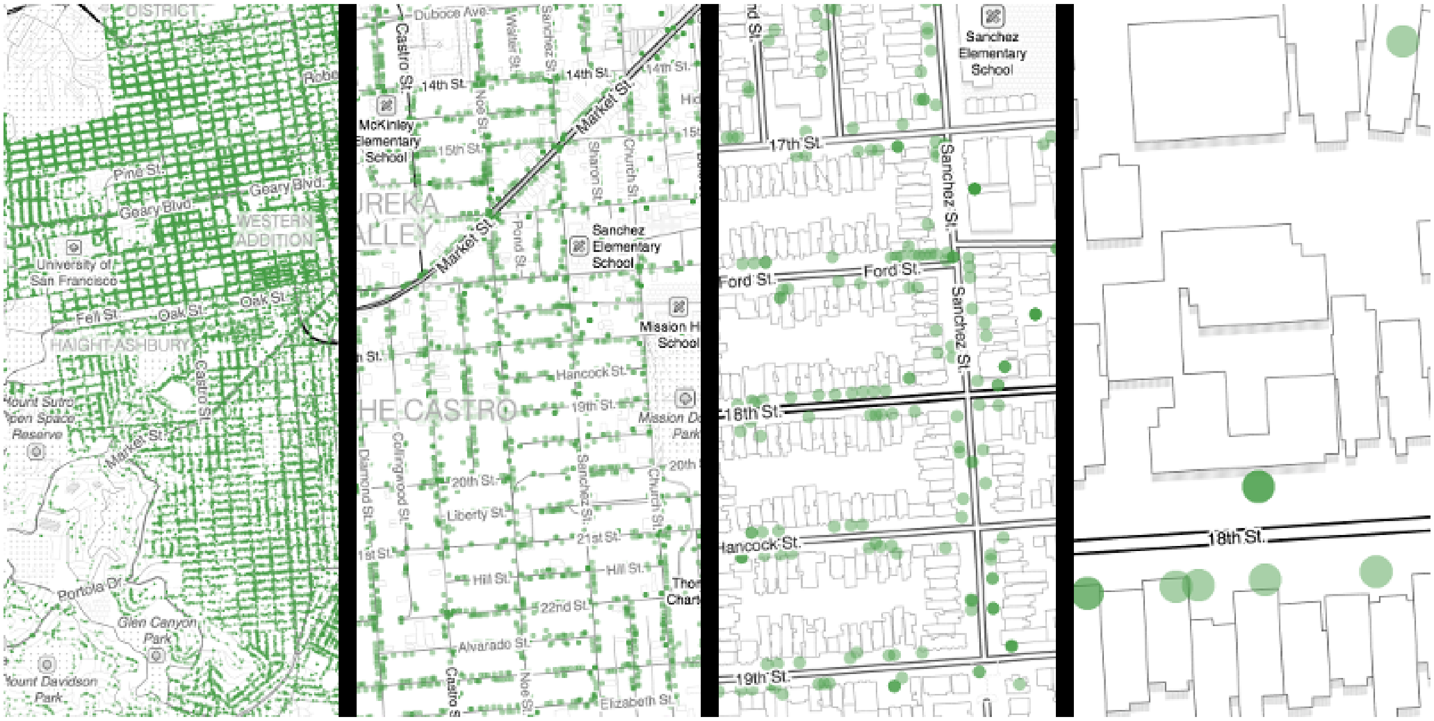
```
        draw:
            points:
                color: [0.243, 0.580, 0.243, 0.5]
                size: 3px
                collide: false
```

Ok, that's really hard to see with 3px dots. Let's add some dynamic sizing so that our dots grow as we zoom in.

We'll use **stops (https://mapzen.com/documentation/tangram/yaml/#stops)**, which you may remember from our **Map Sandwich (https://mapzen.com/blog/map-sandwich/)** post. Stops will change the size of the dot according to our current zoom level:

```
        draw:
            points:
                color: [0.243, 0.580, 0.243, 0.5]
                size: [[12, 1px], [15, 5px], [18, 14px], [18, 20px], [22, 52px]]
                collide: false
```

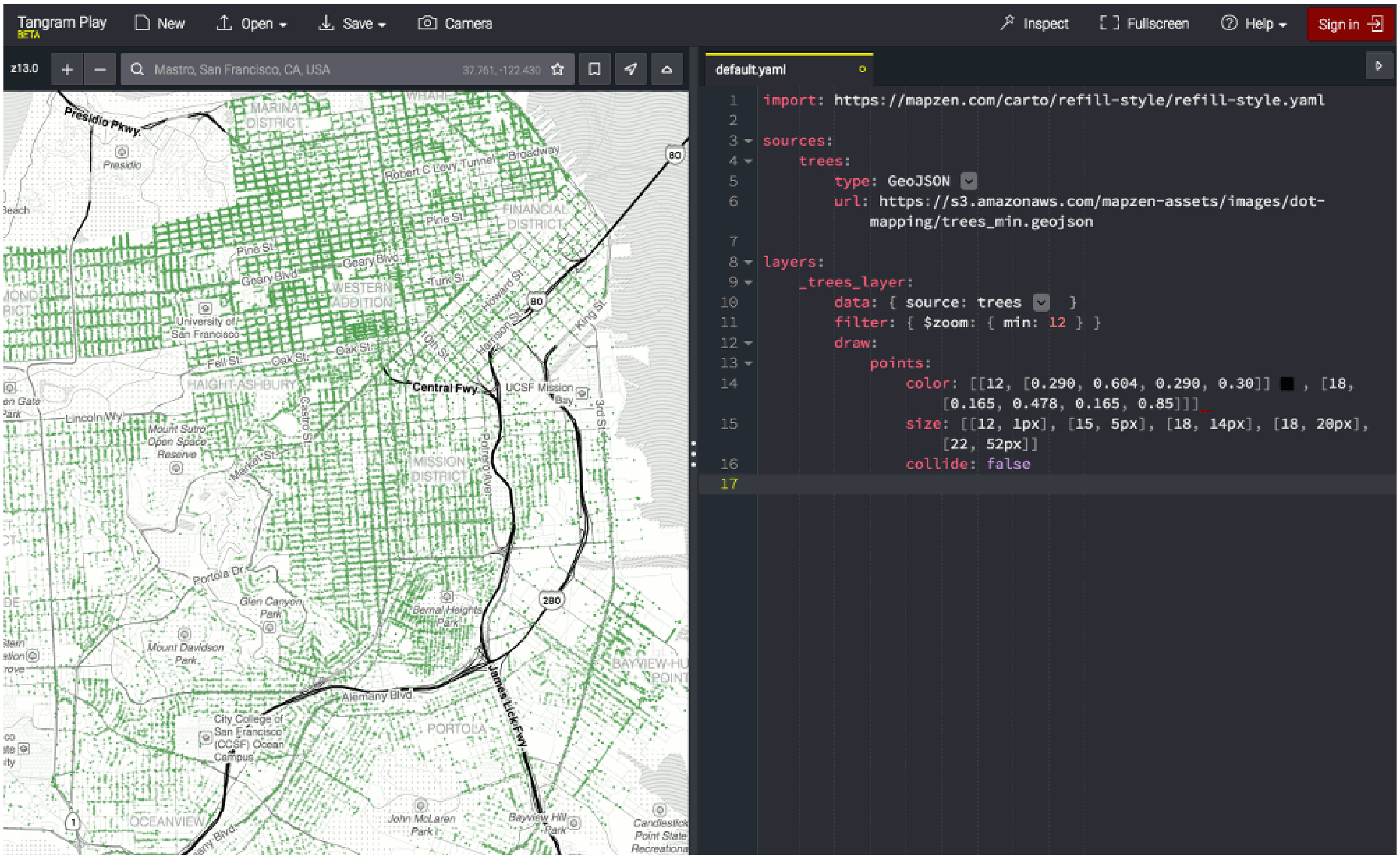
Now, as you zoom in, the dots will grow and you'll be able to see the overlapping trees:


DISTRICT
Pine St.
Geary Blvd.
WESTERN ADDITION
University of San Francisco
Fell St.
Oak St.
HAIGHT-ASHBURY
Castro St.
Market St.
Portola Dr.
Glen Canyon Park
Duboce Ave.
14th St.
McKinley Elementary School
15th St.
Sanchez Elementary School
Hancock St.
THE CASTRO
19th St.
20th St.
Liberty St.
21st St.
Hill St.
22nd St.
Alvarado St.
Elizabeth St.
17th St.
Ford St.
Sanchez St.
18th St.
Hancock St.
19th St.
18th St.

By the way, the two stops at zoom level 18 isn't a typo. I want my dots to jump up in size at z18, rather than continuing the gradual size increase up to that point. Later on we'll add icons to those dots, and the size jump will make more sense then.

We can use this same method for color. We'll make the color a darker, more opaque green the further we zoom in:

```
        draw:
            points:
                color: [[12, [0.290, 0.604, 0.290, 0.30]], [18, [0.165, 0.478, 0.165, 0.8
                size: [[12, 1px], [15, 5px], [18, 14px], [18, 20px], [22, 52px]]
                collide: false
```


Tangram Play
BETA
New
Open
Save
Camera
Inspect
Fullscreen
Help
Sign in
z13.0
Mastro, San Francisco, CA, USA
37.761, -122.430
default.yaml
Presidio Pkwy
Presidio
MARINA DISTRICT
Broadway
Robert C Levy Tunnel
FINANCIAL DISTRICT
Pine St
Geary Blvd
WESTERN ADDITION
Turk St
University of San Francisco
Fell St
Oak St
HAIGHT-ASHBURY
Central Fwy
UCSF Mission Bay
Lincoln Wy
Mount Sutro Open Space Reserve
Market St
MISSION DISTRICT
Portola Dr
Glen Canyon Park
Bernal Heights Park
Mount Davidson Park
BAYVIEW-HUNTERS POINT
Alemany Blvd
PORTOLA
City College of San Francisco (CCSF) Ocean Campus
OCEANVIEW
John McLaren Park
Bayview Hill Park
James Lick Fwy
Candlestick Point State Recreation
80
280

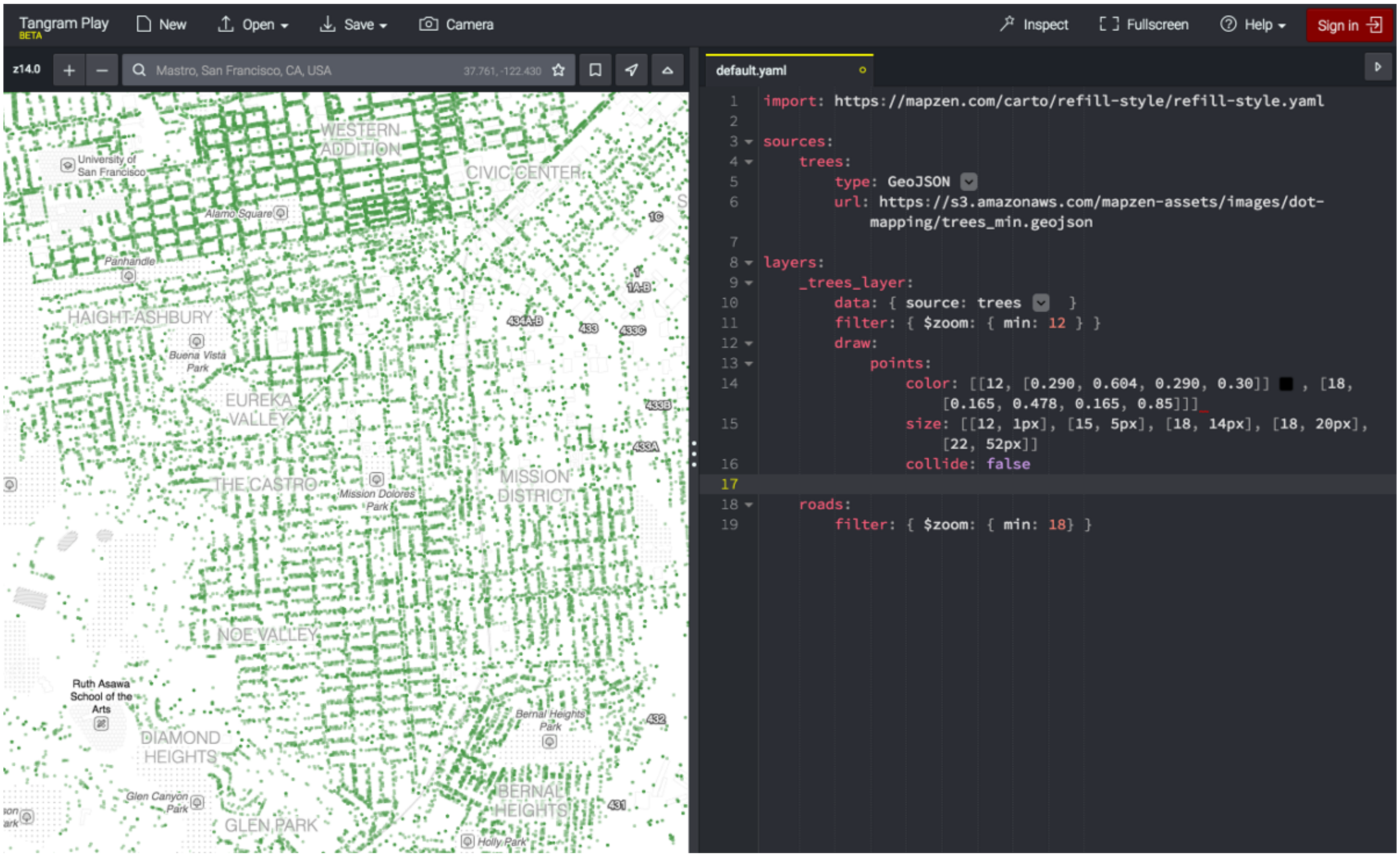
```
import: https://mapzen.com/carto/refill-style/refill-style.yaml

sources:
    trees:
        type: GeoJSON
        url: https://s3.amazonaws.com/mapzen-assets/images/dot-
            mapping/trees_min.geojson

layers:
    _trees_layer:
        data: { source: trees  }
        filter: { $zoom: { min: 12 } }
        draw:
            points:
                color: [[12, [0.290, 0.604, 0.290, 0.30]] , [18,
                    [0.165, 0.478, 0.165, 0.85]]]
                size: [[12, 1px], [15, 5px], [18, 14px], [18, 20px],
                    [22, 52px]]
                collide: false
```

The color change is a subtle difference, but it will help our custom sprites pop when we get to that point.

Let's do one more thing before we move on to sprites. Let's filter out all road lines below zoom 18 by adding this layer to the bottom of our scene file:

```
    roads:
        filter: { $zoom: { min: 18} }
```

You don't need to worry about specifying a data source for this layer, because it's already defined in our imported Refill style. All we're doing here is overriding the filter for that previously defined `roads` layer.


Tangram Play
BETA
New
Open
Save
Camera
Inspect
Fullscreen
Help
Sign in
z14.0
Mastro, San Francisco, CA, USA
37.761, -122.430
default.yaml
WESTERN ADDITION
CIVIC CENTER
University of San Francisco
Alamo Square
Panhandle
HAIGHT-ASHBURY
Buena Vista Park
EUREKA VALLEY
THE CASTRO
Mission Dolores Park
MISSION DISTRICT
NOE VALLEY
Ruth Asawa School of the Arts
DIAMOND HEIGHTS
Glen Canyon Park
GLEN PARK
Bernal Heights Park
BERNAL HEIGHTS
Holly Park

```yaml
import: https://mapzen.com/carto/refill-style/refill-style.yaml

sources:
    trees:
        type: GeoJSON
        url: https://s3.amazonaws.com/mapzen-assets/images/dot-
            mapping/trees_min.geojson

layers:
    _trees_layer:
        data: { source: trees   }
        filter: { $zoom: { min: 12 } }
        draw:
            points:
                color: [[12, [0.290, 0.604, 0.290, 0.30]] , [18,
                    [0.165, 0.478, 0.165, 0.85]]]
                size: [[12, 1px], [15, 5px], [18, 14px], [18, 20px],
                    [22, 52px]]
                collide: false

    roads:
        filter: { $zoom: { min: 18} }
```

Look, Ma. No lines!

Even with the roads now turned off at low zoom levels, our dense tree data gives a pretty clear picture of where streets run in San Francisco.

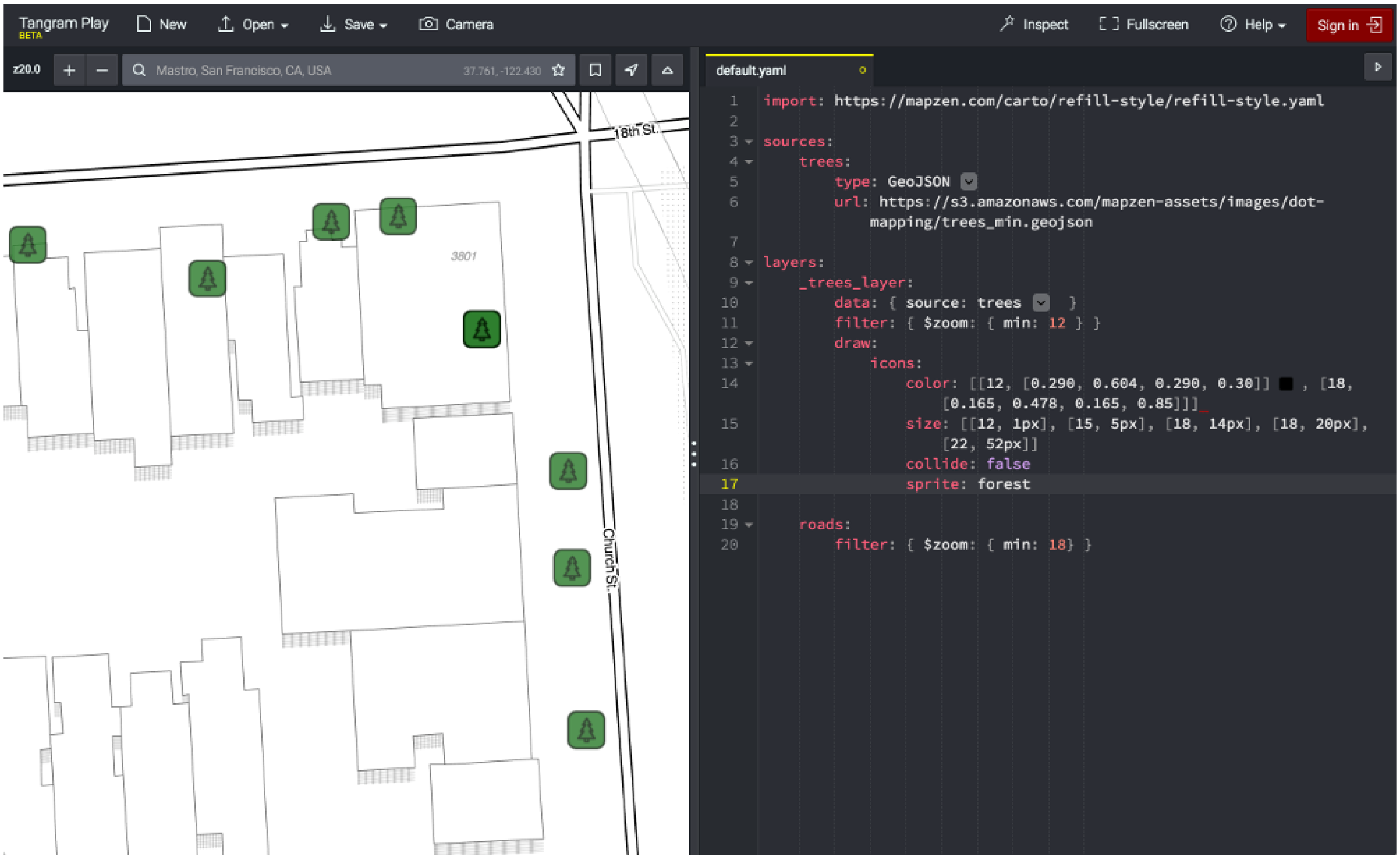
Built-in Sprites

Sprites are simply graphics that we can use to represent a location. One way we can do this is by using the sprites that are baked into Mapzen's basemap styles.

For example, Refill has a custom style called `icons`, which comes with a set of named sprites. Refill uses the `icons` style to represent POIs, but we could hijack it for our own data.

Instead of using the built-in `points` draw style, try using the `icons` style and referencing one of its existing sprites (a sprite called `forest`).

```
        draw:
            icons:
                color: [[12, [0.290, 0.604, 0.290, 0.30]], [18, [0.165, 0.478, 0.165, 0.8
                size: [[12, 1px], [15, 5px], [18, 14px], [18, 20px], [22, 52px]]
                collide: false
                sprite: forest
```


Tangram Play
BETA
New
Open
Save
Camera
Inspect
Fullscreen
Help
Sign in
z20.0
Mastro, San Francisco, CA, USA
37.761, -122.430
18th St.
3801
Church St.
default.yaml

```
import: https://mapzen.com/carto/refill-style/refill-style.yaml

sources:
    trees:
        type: GeoJSON
        url: https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/trees_min.geojson

layers:
    _trees_layer:
        data: { source: trees  }
        filter: { $zoom: { min: 12 } }
        draw:
            icons:
                color: [[12, [0.290, 0.604, 0.290, 0.30]] , [18, [0.165, 0.478, 0.165, 0.85]]]
                size: [[12, 1px], [15, 5px], [18, 14px], [18, 20px], [22, 52px]]
                collide: false
                sprite: forest

    roads:
        filter: { $zoom: { min: 18} }
```

Can't believe I haven't made a single joke about forest sprites.

There are a lot of sprites baked into Refill: airport, aquarium, bench, candy, deli, ferry, kiosk, peak, manor, museum, ruins, school, subway-entrance, winery, zoo, and many many more. Check out our **recently updated icon library (https://mapzen.com/documentation/cartography/icons/#sprites)** for a full list or read on for how to set up your own custom icons.

Custom Icons

Let's go back to our green dots:

```
        draw:
            points:
                color: [[12, [0.290, 0.604, 0.290, 0.30]], [18, [0.165, 0.478, 0.165, 0.8
                size: [[12, 1px], [15, 5px], [18, 14px], [18, 20px], [22, 52px]]
                collide: false
```

I want to turn my green dots into tree icons as I zoom in. To do this, I'm going to layer this **white icon (https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/tree_white.png)** over the green dots. Using an icon with a transparent background like this means that if I decide to change the color of my dots, I won't have to update the color of a static image, as well.

You may remember from a **previous post (https://mapzen.com/blog/map-sandwich/)** that **layering = feature ordering = tricky business**.

There are a couple of things to consider when determining **feature ordering (https://mapzen.com/documentation/vector-tiles/layers/#feature-ordering)** in Tangram. The first is the type of **blend mode (https://mapzen.com/documentation/tangram/styles/#blend)** used by your draw style. We're using the `points` draw style, which has a default blend value of `overlay`. The `overlay` mode is helpful in that it allows us to use alpha values (i.e. transparency). However, in this case it is less helpful in that it simply adds our data to the top of all other layers, regardless of the `order` parameter that we set.

We can get around this in one of two ways:

#1 Change our blend mode to `inlay`, which allows transparency but also honors ordering. Then we can add an `order` parameter following this **set of global ordering variables (https://mapzen.com/documentation/cartography/api-reference/#data-visualization)**.

~ or ~

#2 We can set a **blend_order (https://mapzen.com/documentation/tangram/styles/#blend_order)** parameter on our draw style. The `blend_order` controls the order in which styles are rendered. Styles with a greater `blend_order` value will be drawn on top.

Let's go with #2 for no reason other than we can do this in fewer lines of code.

Near the top of your scene file, let's add a custom style we can use for our tree icons:

```
styles:
    _tree_icons_style:
        base: points
        blend_order: 1
        texture: https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/tree_white.png
```

Our new style (`_tree_icons_style`) will be based on the `points` style with just two small changes:

First, we set a `blend_order` of anything higher than 0. (Yep, even a `1` will get the job done.)

Next, we'll point to the image we want to use for our sprite. We do this using the **texture (https://mapzen.com/documentation/tangram/textures/#sprites)** parameter.

Textures (https://mapzen.com/documentation/tangram/textures/#texture) are far more than just an image reference. And if you're interested, I recommend reading up on the different ways textures can be used in Tangram in the **Tangram Materials Overview (https://mapzen.com/documentation/tangram/Materials-Overview/)**.

For now (and at the risk of upsetting 3D modelers and designers everywhere), you can think of `texture` as simply the path to our icon image.

Let's display our icons in a sublayer of `_trees_layer` called `_trees_icon_sublayer` . This way, we can filter the icons so that they only show up at zoom level 18 and above:

```
        draw:
            points: ...
        _trees_icon_sublayer:
            filter: { $zoom: { min: 18 } }
```

Next, we'll draw the icons using the new `_tree_icons_style` we just added. At this point, we'll also set a dynamic size (keeping them slightly smaller than our green dots) and set `collide` to `false` so that all points show up.

```
        draw:
            points: ...
        _trees_icon_sublayer:
            filter: { $zoom: { min: 18 } }
            draw:
                _tree_icons_style:
                    size: [[18, 14px], [22, 40px]]
                    collide: false
```

And that’s it! You’ll now have a custom tree sprite that shows up when you zoom in past z18.

© Mapzen, OpenStreetMap, and others | Leaflet

Yeah, this is an interactive demo. You can zoom in and out.

Let’s add one last thing to our tree map:

Labels

We first touched on labels in our **Put A Label On It (https://mapzen.com/blog/tangram-labels/)** post, but that was in the context of adding labels to polygons. Adding labels to points isn't so different, except for the fact that you'll need to pay closer attention to collisions. Tangram is careful to avoid collisions between `points` draw styles. We got around that with our tree points by turning off the **collision check (https://mapzen.com/documentation/tangram/draw/#collide)** (`collide: false`).

This time, we'll handle collisions by letting Tangram determine the best placement for the label at various positions around the point.

Yeah, Tangram's smart like that. :)

But there's one catch. We can't use the ***text draw style (https://mapzen.com/documentation/tangram/Styles-Overview/#text)*** we used in our **Put A Label On It (https://mapzen.com/blog/tangram-labels/)** post because `text` draw styles also collide with `points` draw styles. Instead, we'll be using a ***text block (https://mapzen.com/documentation/tangram/draw/#text)*** within the `points` draw style to "attach" the text to the point.

We *could*, then, add a `text` block under our existing `points` draw style in the `_trees_layer` . However, because I want to use the same filters as our `_trees_icon_sublayer` (i.e. only appear at z18 and above), I'm going to add our labels to the icon sublayer.

So, under our `_trees_icon_sublayer` , let's add a second draw style called `points` , to which we can add a `text` block:

```
        _trees_icon_sublayer:
            filter: { $zoom: { min: 18 } }
            draw:
                _tree_icons_style:
                    size: [[18, 14px], [22, 40px]]
                    collide: false
                points:
                    text: ...
```

Next, we'll set our `text_source` (a portion of the `qSpecies` feature property, which I found using the *Inspect Mode* in Tangram Play) and a `font` style:

```
        _trees_icon_sublayer:
            filter: { $zoom: { min: 18 } }
            draw:
                _tree_icons_style:
                    size: [[18, 14px], [22, 40px]]
                    collide: false
                points:
                    text:
                        text_source: |
                            function () {
                                return feature.qSpecies.split(':: ')[1];
                            }
                        font:
                            fill: forestgreen
                            size: 10pt
                            stroke:
                                width: 5px
                                color: white
```

Finally, to ensure that our icons continue to appear even if there's no space for a label, we'll also add the `optional: true` parameter to our `text` block. If we were to remove that parameter, any point that cannot display a label (due to collisions) would hide both that label and the icon.

```
        _trees_icon_sublayer:
            filter: { $zoom: { min: 18 } }
            draw:
                _tree_icons_style:
                    size: [[18, 14px], [22, 40px]]
                    collide: false
                points:
                    text:
                        text_source: |
                            function () {
                                return feature.qSpecies.split(':: ')[1];
                            }
                        font:
                            fill: forestgreen
                            size: 10pt
                            stroke:
                                width: 5px
                                color: white
                        optional: true
```

And there you have it. Labels!

By the way, if you'd rather have more control over the placement of your labels, check out the `anchor` and `offset` parameters listed under Tangram's **text behavior documentation (https://mapzen.com/documentation/tangram/draw/#text-behavior)**. That also happens to be a great resource for better understanding how `points` and `text` relate to one another.

© Mapzen, OpenStreetMap, and others | Leaflet

Zoom in to see additional labels appear.

The full code for this exercise can be seen on **bl.ocks.org (https://bl.ocks.org/rfriberg/5f031f0fe81306cd3c18e07cc1d1c85a)**. You can also view and modify the final scene file using **Tangram Play (https://mapzen.com/tangram/play/?scene=https://s3.amazonaws.com/mapzen-assets/images/dot-mapping/scene_final.yaml#13.0375/37.7614/-122.4296)**.

An **interactive version (https://bl.ocks.org/rfriberg/a33746362b414929bf701c5dd7f39bad)** of this map is also available. (Since San Francisco loves pop-ups, we added a few. Some trees have a surprise.)

If you have questions or want to show off something you've made with Mapzen, **drop us a line (mailto:hello@mapzen.com)**. We love to hear from you!

Dots image courtesy of the ***Tootsie Company (http://www.tootsie.com/gal_wallpaper.php)*** *via* ***Business Insider (http://www.businessinsider.com/the-15-best-and-worst-halloween-candy-2013-10)****.*

~~~

Check out additional tutorials from the **Make Your Own (https://mapzen.com/tag/make-your-own/)** series:

- **One Minute Map (https://mapzen.com/blog/one-minute-map/)**
- **Map Sandwich (https://mapzen.com/blog/map-sandwich/)**
- **Filters & Functions (https://mapzen.com/blog/filters-and-functions/)**
- **Put A Label On It (https://mapzen.com/blog/tangram-labels/)**
- **Interactive Mapping with Tangram (https://mapzen.com/blog/tangram-interactivity/)**
- **Lots of Dots (https://mapzen.com/blog/lots-of-dots/)**
- **Customize Your Search (https://mapzen.com/blog/customize-your-search/)**

· 05 January 2017 ·

Rhonda Friberg

Rhonda is a javascripter who makes maps, trouble, and the occasional pie.

© 2017 Mapzen
~~~

All roads lead to Washington, D.C.

mobility (/tag/mobility) routing (/tag/routing) transitland (/tag/transitland)

Mapzen will be in Washington, D.C. next week for the country's largest gathering of "transpo" enthusiasts and professionals. (BTW, that's "transpo" for transportation.)

We're looking forward to sharing **our fully open-source and open-data approach to mobility (/blog/introducing-mapzen-mobility)**, announcing our collaboration with the World Bank and partners on the **Open Traffic platform (/blog/announcing-open-traffic/)**, and meeting you.

Come find us at:

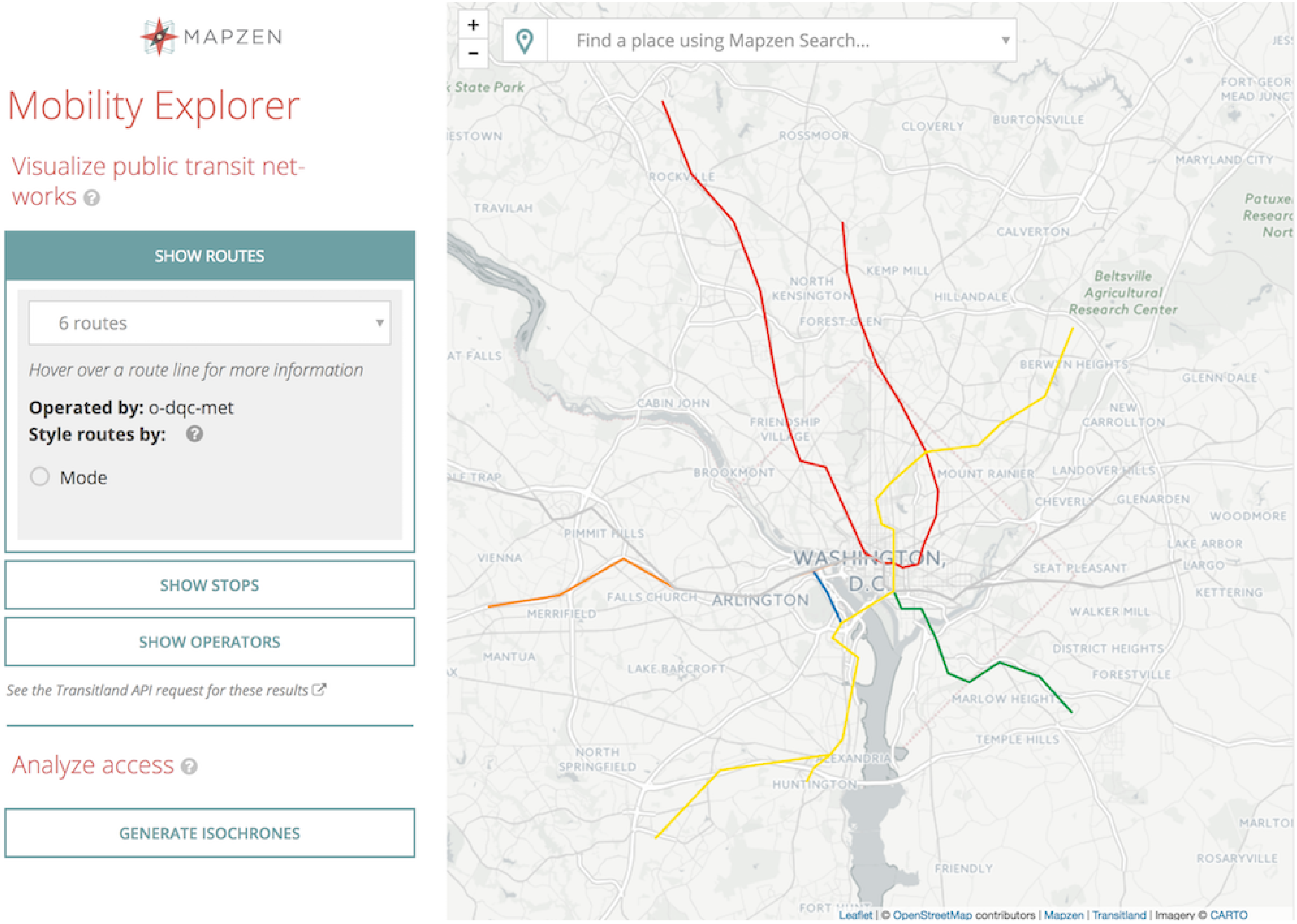
- ***Transportation Camp*** (Saturday, Jan. 7): Dave Nesbitt, Meghan Hade, and Greg Knisely will be on hand to present **Mapzen Mobility Explorer (/blog/introducing-mobility-explorer/)**, **Turn-by-Turn (/products/turn-by-turn/)**, **Isochrone (/blog/introducing-isochrone-service/)**, and new functionality in all three services. **Registration required (http://transportationcamp.org/events/dc-2017/)**.

- ***Mapping + Transpo Happy Hour*** (Sunday, Jan. 8): For the third year in a row, we're co-sponsoring a happy hour with our friends at Conveyal and TransitScreen. **Please RSVP (https://trbparty2017.splashthat.com/)** and join us for drinks and appetizers.

- ***Transportation Techies meet-up*** (Tuesday, Jan. 10): Drew Dara-Abrams will be presenting our latest additions to Mapzen Turn-by-Turn and Isochrone to support bicycle routing and analysis. **Free with an RSVP (https://www.meetup.com/Transportation-Techies/events/235892038/)**.

- ***World Bank and WRI Transforming Transportation conference*** (Friday, Jan. 13): Alyssa Wright and Drew Dara-Abrams will be on hand to answer questions about Mapzen Mobility and Open Traffic, and Alyssa will be speaking on the power of open data, software, and communities for transportation. **Registration required (http://www.transformingtransportation.org/)**.

Can't make it to D.C. in person? You can still **"ride" the Metro using Mapzen Mobility Explorer (https://mapzen.com/mobility/explorer/#/routes?bbox=-77.29774475097656%2C38.725161847874716%2C-76.48269653320312%2C39.16946468278697&operated_by=o-dqc-met&vehicle_type=metro)**:



6 routes

Hover over a route line for more information

Operated by: o-dqc-met

Style routes by:

○ Mode

See the Transitland API request for these results

MAPZEN
Mobility Explorer
Visualize public transit networks
SHOW ROUTES
SHOW STOPS
SHOW OPERATORS
Analyze access
GENERATE ISOCHRONES
+
−
Find a place using Mapzen Search...
WASHINGTON, D.C.
ARLINGTON
ROCKVILLE
NORTH KENSINGTON
FOREST GLEN
BERWYN HEIGHTS
MOUNT RAINIER
FALLS CHURCH
MERRIFIELD
VIENNA
ALEXANDRIA
HUNTINGTON
NORTH SPRINGFIELD
MARLOW HEIGHTS
FRIENDLY
Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery © CARTO

(https://mapzen.com/mobility/explorer/#/routes?bbox=-77.29774475097656%2C38.725161847874716%2C-76.48269653320312%2C39.16946468278697&operated_by=o-dqc-met&vehicle_type=metro)

Photo of DC Metro station thanks to ***RJ Schmidt (https://www.flickr.com/photos/rj_schmidt/5557928950/in/photolist-9t8QDd-FLnco-9t8QHj-5ni4zx-dKj9N9-dpJNhU-a2H5Kb-o7Vhtv-cvsCoS-btYp74-5nnkdo-CzTcy-e1ykZD-dfNtKU-CiwmA-9xeHxV-3DxPED-9CxnPT-5xprsQ-2t2xq-2X33F-2t2zm-7ocTcb-btYpCa-8xtsWw-fqymwe-jum24W-bzNx8U-YWMc-btYpnk-6cea5D-dmVsVC-r4v5DQ-b9Y1fe-btYq4z-4a3LuV-KZQMj-4ERtRK-9mXVZs-5gihoH-bMvabV-cauV71-6zwUi-82KV22-JMq3h-9Tdk7d-5mPqKX-6ip5Cw-6dnSh4-juiYQc)***.

· 06 January 2017 ·

Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

Alyssa Wright

Alyssa Wright does community where the phone and meeting are her superpowers of choice.

David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

Greg Knisely

Greg is a data munger and software engineer for Mapzen's routing software.

Meghan Hade

Meghan is a software engineer with a background in urban planning on Mapzen's Mobility team. She works in javascript and plays in calligraphy, block printing, and finding ways to stash n+1 bikes in a San Francisco apartment.

© 2017 Mapzen

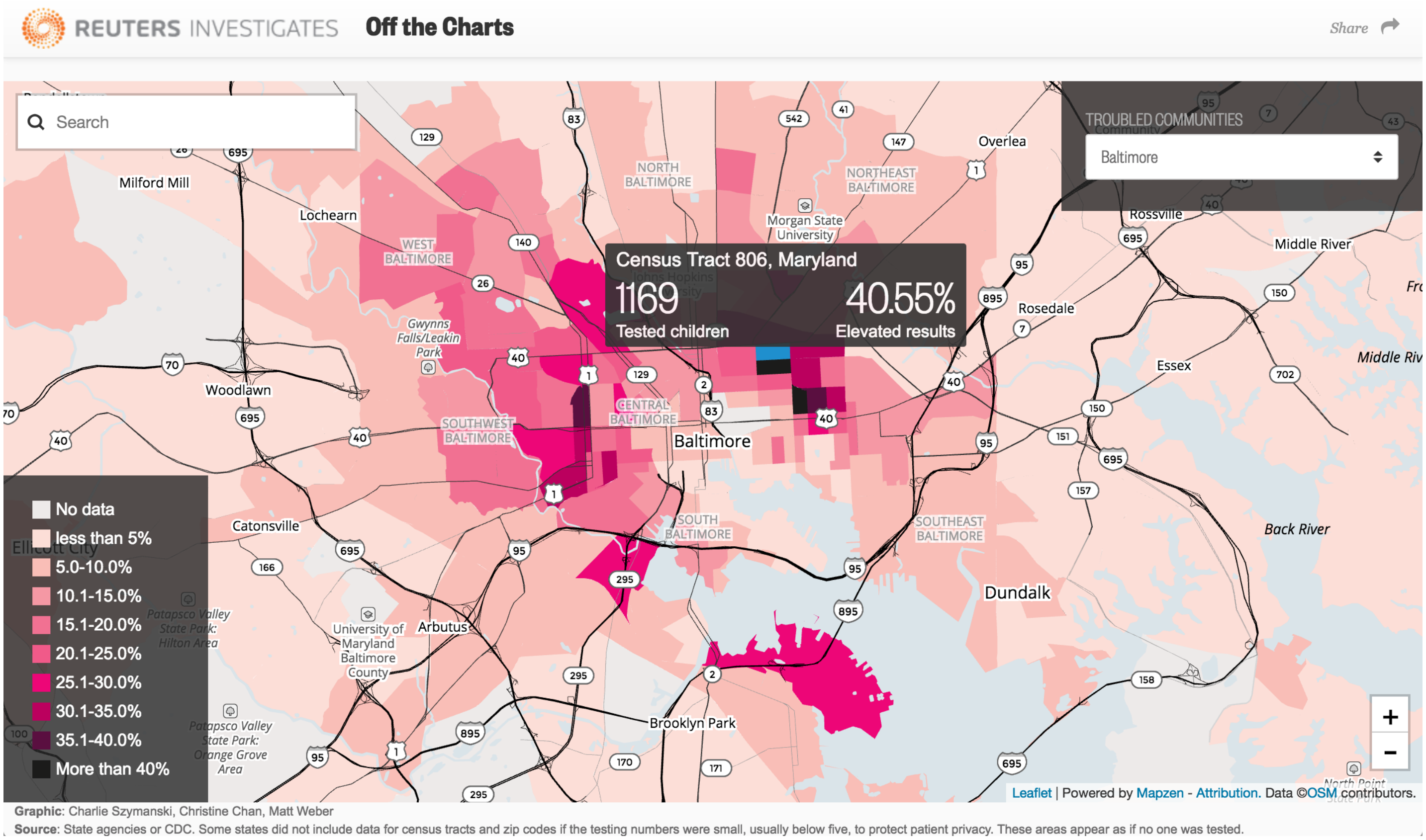
Made With Mapzen

tangram (/tag/tangram) data (/tag/data) mobility (/tag/mobility) terrain (/tag/terrain) vector-tiles (/tag/vector-tiles) cartography (/tag/cartography) search (/tag/search)

We’re happy to see lots of different groups using tools and data from Mapzen – here are a few examples from the past month.

Reuters

Reuters used Tangram to **show 3,000 spots in the United States where lead poisoning is worse than Flint, Michigan (http://reut.rs/2hZEBIS)**.

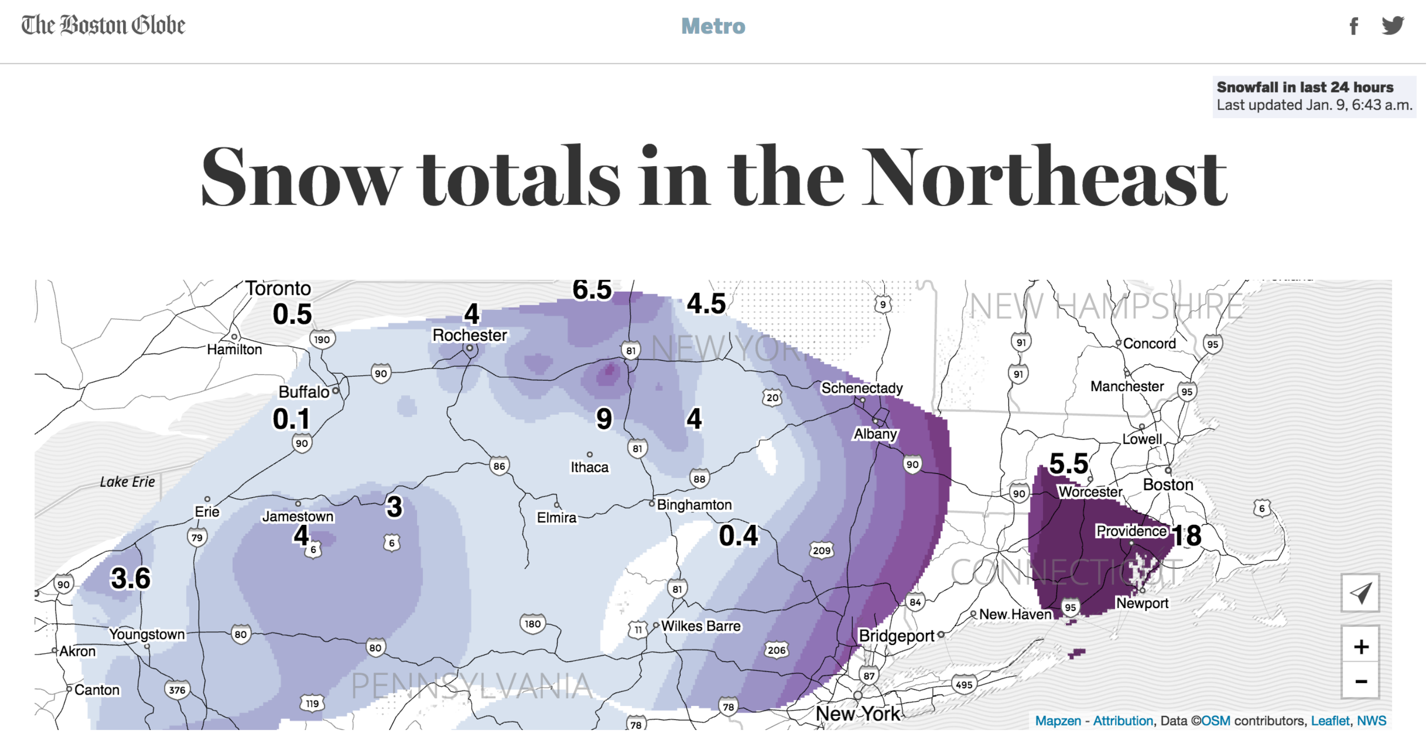

REUTERS INVESTIGATES
Off the Charts
Share
Search
TROUBLED COMMUNITIES
Baltimore
Census Tract 806, Maryland
1169
Tested children
40.55%
Elevated results
No data
less than 5%
5.0-10.0%
10.1-15.0%
15.1-20.0%
20.1-25.0%
25.1-30.0%
30.1-35.0%
35.1-40.0%
More than 40%
Milford Mill
Lochearn
WEST BALTIMORE
NORTH BALTIMORE
NORTHEAST BALTIMORE
Morgan State University
Overlea
Rossville
Middle River
Rosedale
Essex
Woodlawn
SOUTHWEST BALTIMORE
CENTRAL BALTIMORE
Baltimore
Catonsville
SOUTH BALTIMORE
SOUTHEAST BALTIMORE
Dundalk
Back River
Arbutus
University of Maryland Baltimore County
Brooklyn Park
Gwynns Falls/Leakin Park
Patapsco Valley State Park: Hilton Area
Patapsco Valley State Park: Orange Grove Area
Ellicott City
+
−
Leaflet | Powered by Mapzen - Attribution. Data ©OSM contributors.

Graphic: Charlie Szymanski, Christine Chan, Matt Weber

Source: State agencies or CDC. Some states did not include data for census tracts and zip codes if the testing numbers were small, usually below five, to protect patient privacy. These areas appear as if no one was tested.

The Boston Globe

It’s been quite a winter and The Boston Globe is using Mapzen tiles and Tangram **to show how much snow has fallen in the Northeast (https://apps.bostonglobe.com/metro/graphics/2016/12/snow-totals/)**.



The Boston Globe

Metro

Snowfall in last 24 hours
Last updated Jan. 9, 6:43 a.m.

Snow totals in the Northeast

Toronto
0.5
Hamilton
6.5
4
Rochester
4.5
NEW YORK
NEW HAMPSHIRE
Concord
Manchester
Lowell
Buffalo
0.1
9
4
Schenectady
Albany
Ithaca
5.5
Worcester
Boston
Lake Erie
Erie
Jamestown
3
4
Elmira
Binghamton
0.4
Providence
18
3.6
CONNECTICUT
New Haven
Newport
Youngstown
Akron
Canton
Wilkes Barre
Bridgeport
PENNSYLVANIA
New York
+
-
Mapzen - Attribution, Data ©OSM contributors, Leaflet, NWS

Los Angeles Times

The LA Times is using Tangram to **quickly generate maps for breaking news stories (https://twitter.com/latimes/status/812120574713614336)**.

Los Angeles Times
@latimes

Air traffic controller removed from post after directing jetliner toward Mt. Wilson lat.ms/2hwjp9q

6:20 PM - Dec 22, 2016

11 62 40

The Washington Post

Philip Bump of The Fix **used Mapzen Search to help find the closest place that voted the opposite of where you lived (https://www.washingtonpost.com/news/the-fix/wp/2016/11/15/bringing-america-together-find-the-closest-place-that-voted-the-opposite-of-where-you-live/)**.

county, state SEARCH OR FIND ME

You are in/near:

San Francisco County, California

Clinton: 85.3%

Trump: 9.9%

ID
WY
Casper
Cheyenne
Denver
UT
ramento
San Francisco
Fresno
CA
Las Vegas
Flagstaff
Albuquerque
Amari
NM
Santa Barbara
San Bernardino
AZ
Phoenix
Los Angeles
San Diego
Mexicali
Tucson
El Paso
+
−
Mapzen - Attribution, Data ©OSM contributors, Leaflet

Where you can go to see how the other half lives

According to this, you're in **San Francisco County, California**, where Clinton won by 75.4 points..

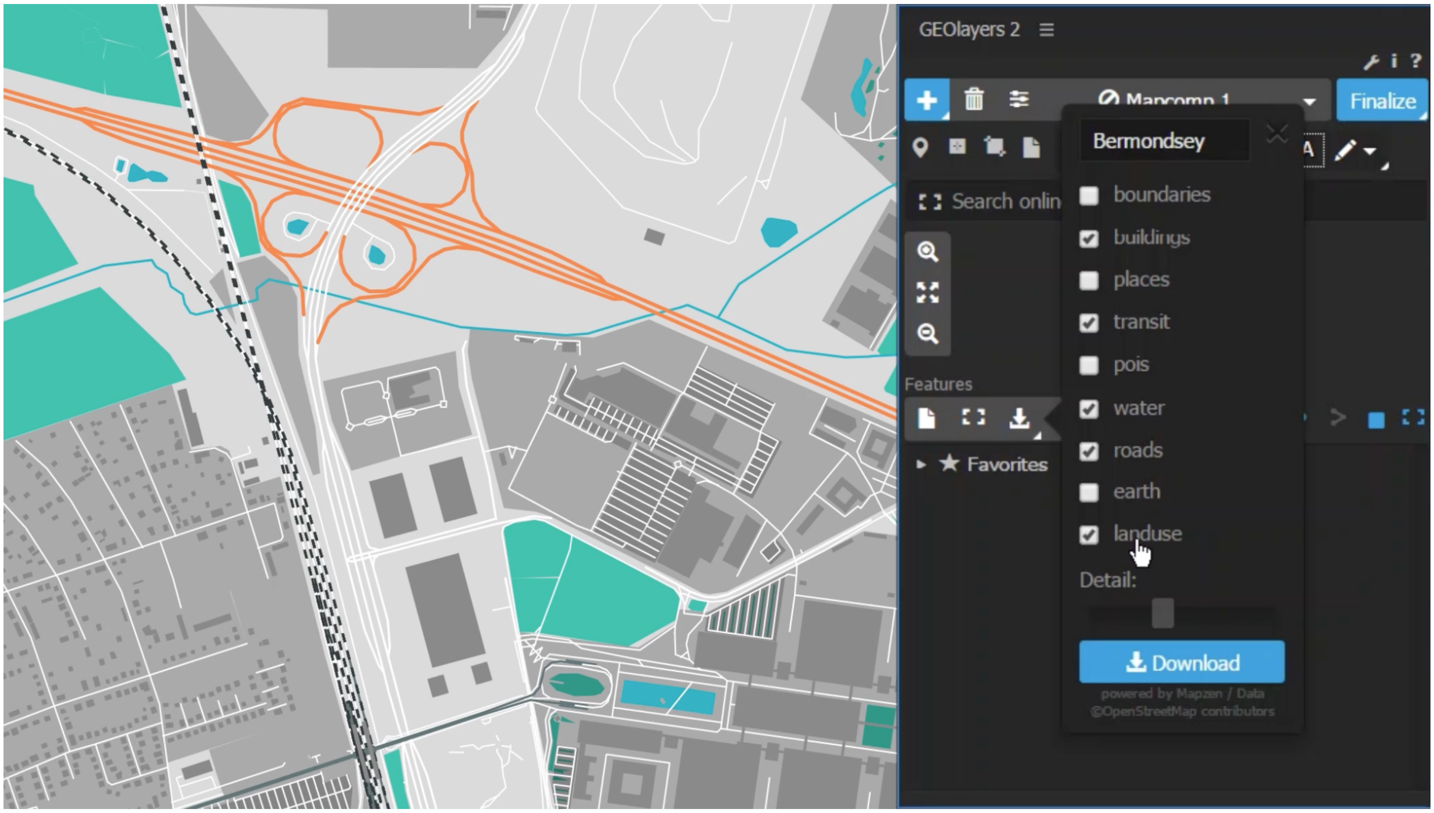
If you want to visit somewhere nearby that held the opposite view of the election, you should travel to **Eureka County, Nevada**.

	County	Clinton	Trump
Where you are	San Francisco County, California	85.3	9.9
Nearest opposite	Eureka County, Nevada	8.7	84.7
Most like	Baltimore city, Maryland	85.4	10.9
Most unlike	Cherry County, Nebraska	10.3	85.1

FYI, nerds: The mapping system is powered by Mapzen, an open source mapping tool.

Getting Maps into AfterEffects

GeoLayers (http://aescripts.com/geolayers/) lets you browse and animate maps in AfterEffects using Mapzen vector data, including roads, buildings, terrain and even routing.


GEOlayers 2
Mapcomp 1
Finalize
Search onlin
Features
Favorites
Download
powered by Mapzen / Data ©OpenStreetMap contributors

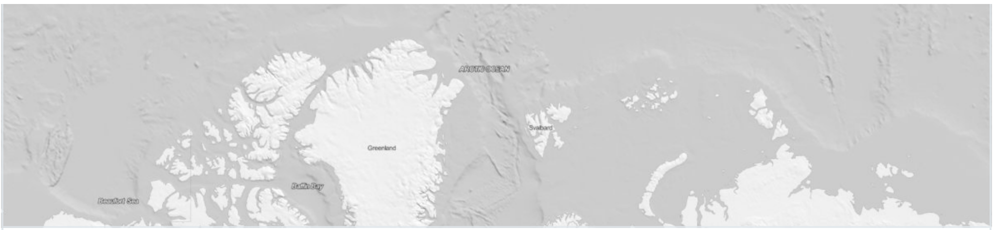
Bermondsey

☐ boundaries
☑ buildings
☐ places
☑ transit
☐ pois
☑ water
☑ roads
☐ earth
☑ landuse

Detail:

Maproom

Maproom (https://www.klaasnienhuis.nl/product/maproom/), an Autodesk plug-in, has a new terrain system that uses Mapzen tiles. This means terrain and bathymetry data any any scale!


ARCTIC OCEAN
Greenland
Svalbard
Baffin Bay
Beaufort Sea

Klaas Nienhuis
@klaasnienhuis

New terrain system using @mapzen data includes #bathymetry. Maproom will soon have terrain data at any scale. #3dsMax

8:13 AM - Dec 30, 2016

7 24

Let us know (mailto:hello@mapzen.com?subject=I%20made%20__%20with%20Mapzen) how you are using Mapzen!

· 13 January 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

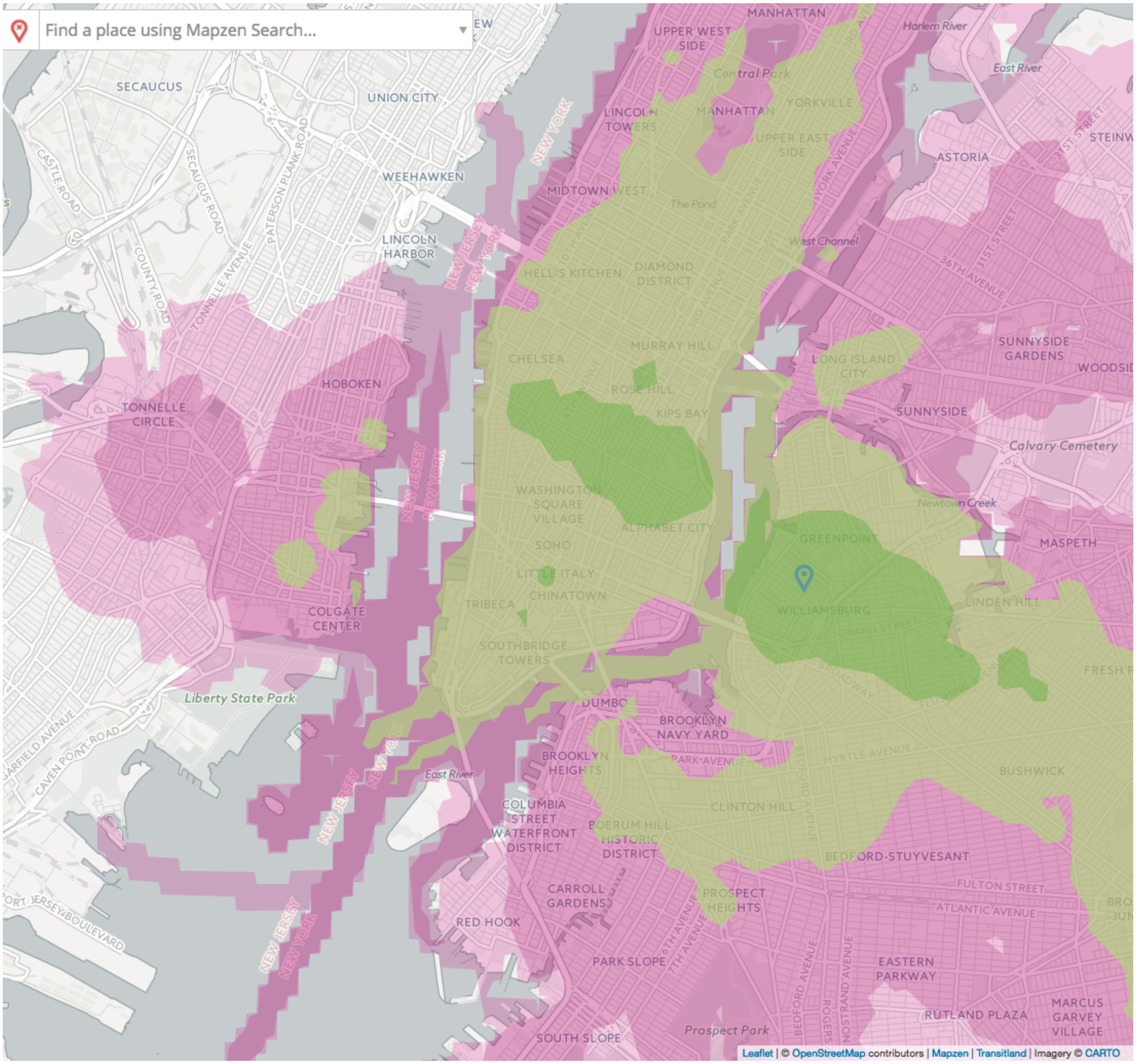
Transit Inclusion & Exclusion

mobility (/tag/mobility)

Tunneling through transit data

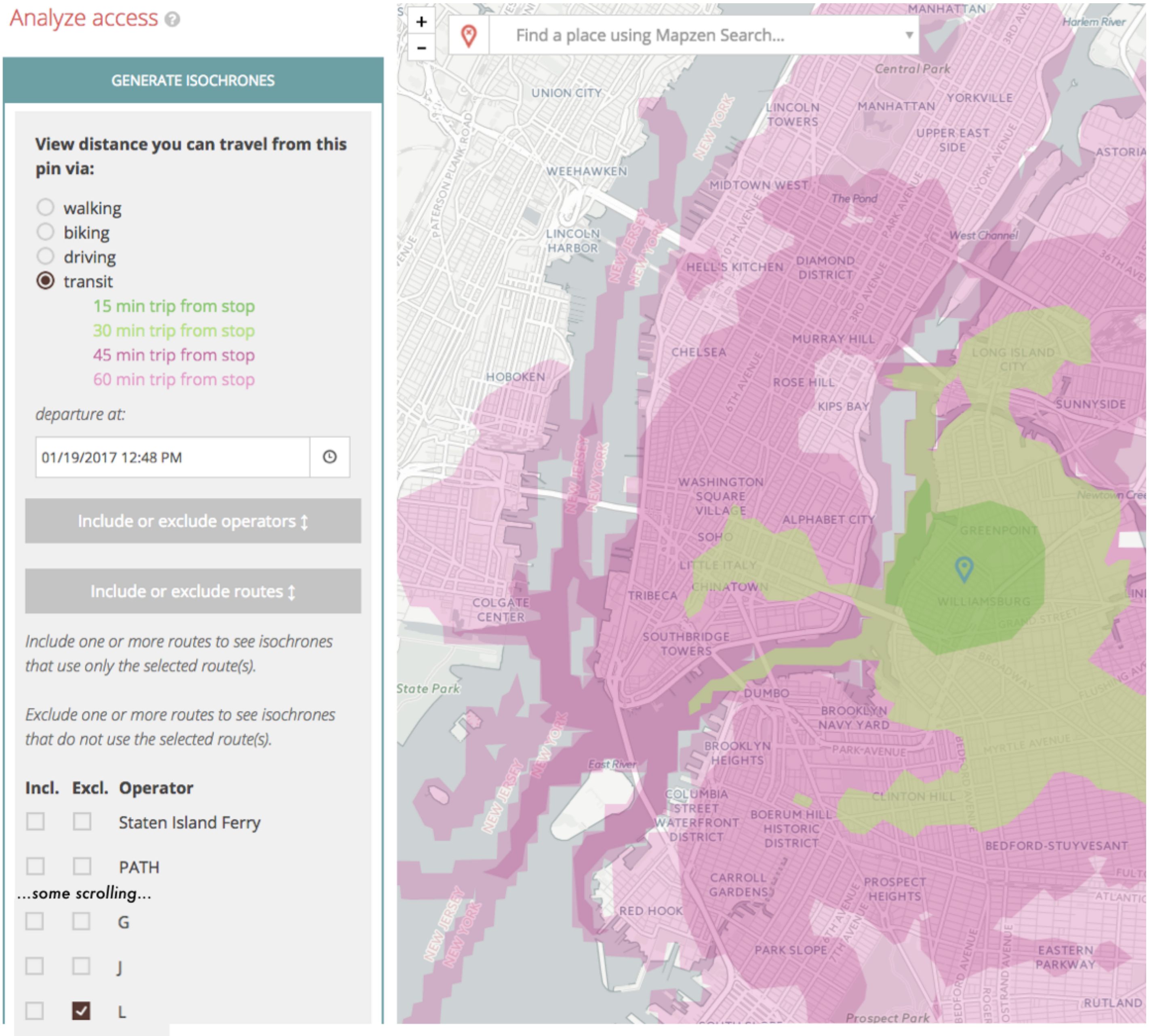
As any New Yorker will tell you, in 2019 **the L will shut down between Manhattan and Brooklyn (http://www.nytimes.com/2016/07/26/nyregion/l-train-will-shut-down-between-manhattan-and-brooklyn-in-2019-for-18-months.html?_r=0)** for a year and a half so the MTA can repair damage in the Canarsie Tunnel from Hurricane Sandy. Our friends at CARTO **conducted a fantastic analysis last year on the impact of the shutdown (https://carto.com/blog/looking-at-the-l)**. We've added some features to our **Mobility API (https://mapzen.com/documentation/mobility/)** and the **Mapzen Mobility Explorer (https://mapzen.com/mobility/explorer/#/?bbox=-89.45145606994629%2C43.026172725579556%2C-89.33326721191406%2C43.11971813686748)** that will make dynamic analysis of transit closures (as well as the addition of new lines) even easier to assess.

With the isochrone service in the Mobility Explorer (https://mapzen.com/mobility/explorer/#/isochrones?bbox=-74.023647,40.684023,-73.87516,40.766502&isochrone_mode=multimodal&pin=40.717304%2C-73.956872), we can see the present level of service for the L from Bedford station in Brooklyn. You will notice that it takes a rider no more than 15 minutes to get to the 8th Ave station in Manhattan.


Find a place using Mapzen Search...
SECAUCUS
UNION CITY
WEEHAWKEN
LINCOLN HARBOR
HOBOKEN
TONNELLE CIRCLE
COLGATE CENTER
Liberty State Park
MANHATTAN
UPPER WEST SIDE
Central Park
YORKVILLE
UPPER EAST SIDE
LINCOLN TOWERS
MIDTOWN WEST
HELL'S KITCHEN
DIAMOND DISTRICT
CHELSEA
MURRAY HILL
ROSE HILL
KIPS BAY
WASHINGTON SQUARE VILLAGE
ALPHABET CITY
SOHO
LITTLE ITALY
CHINATOWN
TRIBECA
SOUTHBRIDGE TOWERS
Harlem River
East River
ASTORIA
STEINW
SUNNYSIDE GARDENS
WOODSID
LONG ISLAND CITY
SUNNYSIDE
Calvary Cemetery
Newtown Creek
MASPETH
GREENPOINT
WILLIAMSBURG
LINDEN HILL
FRESH P
DUMBO
BROOKLYN NAVY YARD
BROOKLYN HEIGHTS
COLUMBIA STREET WATERFRONT DISTRICT
BOERUM HILL HISTORIC DISTRICT
CLINTON HILL
BUSHWICK
BEDFORD-STUYVESANT
CARROLL GARDENS
RED HOOK
PROSPECT HEIGHTS
PARK SLOPE
EASTERN PARKWAY
RUTLAND PLAZA
MARCUS GARVEY VILLAGE
SOUTH SLOPE
Prospect Park
NEW JERSEY
NEW YORK
Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery © CARTO

(dark green, 15 minutes from stop. light green, 30 mins. purple, 45 mins. pink, 60 mins.)

We recently added `include` and `exclude` filters in the Valhalla API. You can filter transit agencies and routes in the Mobility Explorer and change the isochrone on the fly.



Analyze access

GENERATE ISOCHRONES

View distance you can travel from this pin via:

○ walking
○ biking
○ driving
● transit

15 min trip from stop
30 min trip from stop
45 min trip from stop
60 min trip from stop

departure at:

01/19/2017 12:48 PM

Include or exclude operators ↕

Include or exclude routes ↕

Include one or more routes to see isochrones that use only the selected route(s).

Exclude one or more routes to see isochrones that do not use the selected route(s).

Incl.	Excl.	Operator
☐	☐	Staten Island Ferry
☐	☐	PATH

...some scrolling...

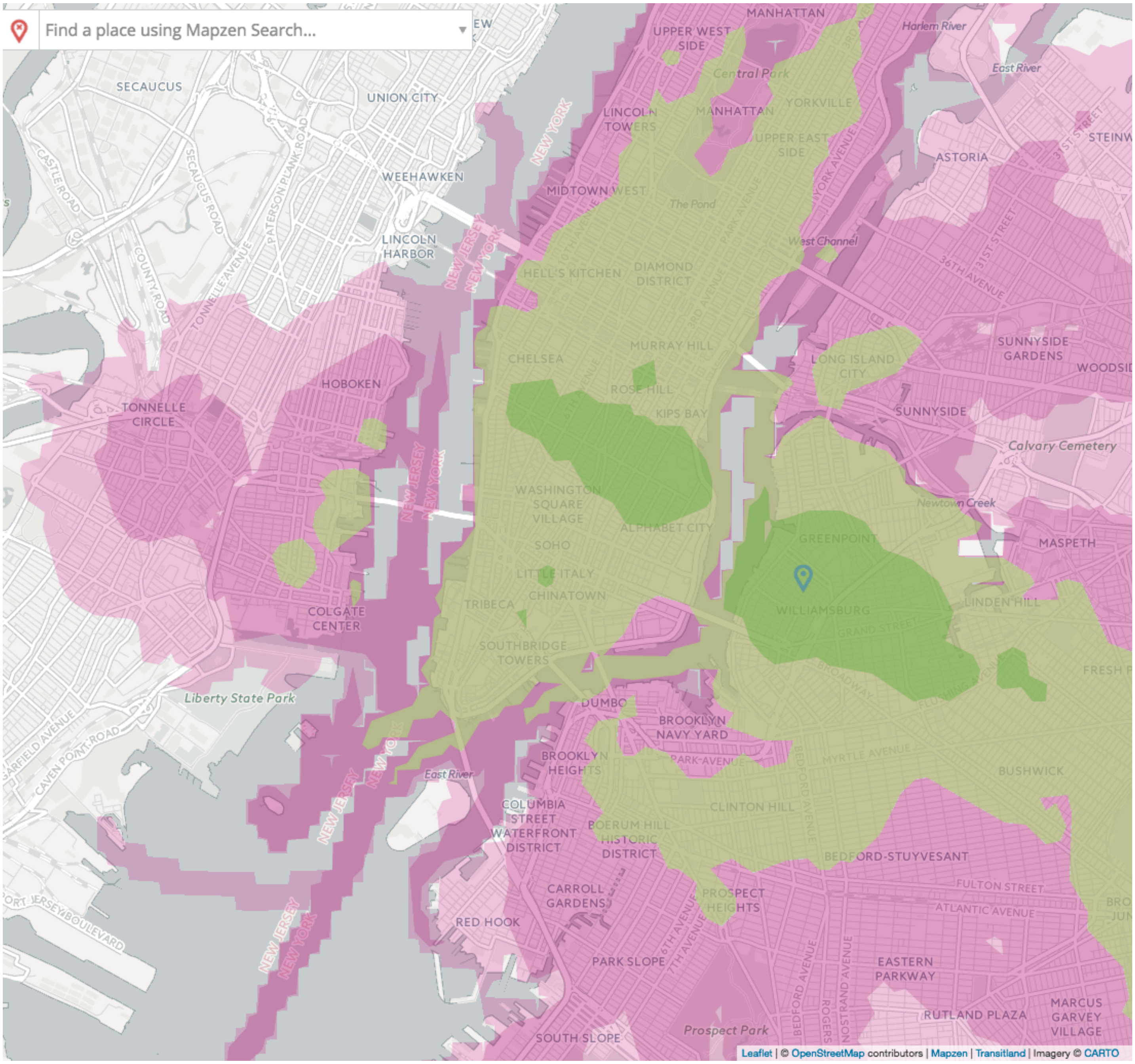
Incl.	Excl.	Operator
☐	☐	G
☐	☐	J
☐	☑	L

+
-
Find a place using Mapzen Search...
MANHATTAN
Harlem River
Central Park
UNION CITY
LINCOLN TOWERS
MANHATTAN
YORKVILLE
UPPER EAST SIDE
ASTORIA
WEEHAWKEN
MIDTOWN WEST
The Pond
LINCOLN HARBOR
West Channel
HELL'S KITCHEN
DIAMOND DISTRICT
NEW JERSEY
NEW YORK
MURRAY HILL
CHELSEA
LONG ISLAND CITY
HOBOKEN
ROSE HILL
KIPS BAY
SUNNYSIDE
WASHINGTON SQUARE VILLAGE
ALPHABET CITY
SOHO
GREENPOINT
LITTLE ITALY
CHINATOWN
TRIBECA
WILLIAMSBURG
COLGATE CENTER
SOUTHBRIDGE TOWERS
State Park
DUMBO
BROOKLYN NAVY YARD
BROOKLYN HEIGHTS
East River
COLUMBIA STREET WATERFRONT DISTRICT
BOERUM HILL HISTORIC DISTRICT
CLINTON HILL
BEDFORD-STUYVESANT
CARROLL GARDENS
PROSPECT HEIGHTS
RED HOOK
PARK SLOPE
EASTERN PARKWAY
RUTLAND
Prospect Park

(For more granular control, you can use the Mobility API to filter individual stops from transit routing results and return isochrones as GeoJSON. Below, we exclude stations from 8th Ave. to Bedford Ave. that will be shut down when the Canarsie Tunnel closes in 2019.)

```
https://matrix.mapzen.com/isochrone?json={"locations":
[{"lat":40.7178,"lon":-73.9575}],"costing":"transit","contours":[{"time":5},
{"time":15},{"time":30},
{"time":45}],"polygons":true,"denoise":0.2,"generalize":150,"costing_options":
{"transit":{"filters":{"stops":{"ids":["s-dr5rswb4zp-1av<l06n","s-dr5rsrhyn3-
3av<l05n","s-dr5rsr9975-unionsq~14st<l03n","s-dr5ru0j0jv-6av<l02n","s-dr5ru02yqu-
8av<l01n"],"action":"exclude"}}}}}
```

The commute time to the 8th Ave station (or most parts of Manhattan) will increase to 45 minutes during the L shutdown. As noted in the **article (http://www.nytimes.com/2016/07/26/nyregion/l-train-will-shut-down-between-manhattan-and-brooklyn-in-2019-for-18-months.html?_r=0)**, during each weekday 225K people will have an increased commute time for almost everywhere within Manhattan.


Find a place using Mapzen Search...
SECAUCUS
UNION CITY
WEEHAWKEN
LINCOLN HARBOR
HOBOKEN
TONNELLE CIRCLE
COLGATE CENTER
Liberty State Park
MANHATTAN
UPPER WEST SIDE
Central Park
YORKVILLE
UPPER EAST SIDE
LINCOLN TOWERS
MIDTOWN WEST
HELL'S KITCHEN
DIAMOND DISTRICT
CHELSEA
MURRAY HILL
ROSE HILL
KIPS BAY
WASHINGTON SQUARE VILLAGE
ALPHABET CITY
SOHO
LITTLE ITALY
CHINATOWN
TRIBECA
SOUTHBRIDGE TOWERS
Harlem River
East River
ASTORIA
STEINW
SUNNYSIDE GARDENS
WOODSID
LONG ISLAND CITY
SUNNYSIDE
Calvary Cemetery
Newtown Creek
MASPETH
GREENPOINT
WILLIAMSBURG
LINDEN HILL
FRESH P
DUMBO
BROOKLYN NAVY YARD
BROOKLYN HEIGHTS
COLUMBIA STREET WATERFRONT DISTRICT
BOERUM HILL HISTORIC DISTRICT
CLINTON HILL
BUSHWICK
BEDFORD-STUYVESANT
CARROLL GARDENS
PROSPECT HEIGHTS
RED HOOK
PARK SLOPE
EASTERN PARKWAY
RUTLAND PLAZA
MARCUS GARVEY VILLAGE
SOUTH SLOPE
Prospect Park
NEW JERSEY
NEW YORK
Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery © CARTO

(dark green, 15 minutes from stop. light green, 30 mins. purple, 45 mins. pink, 60 mins.)

You can investigate **travel times excluding the L (https://mapzen.com/mobility/explorer/#/isochrones?bbox=-74.023647%2C40.684023%2C-73.87516%2C40.766502&isochrone_mode=multimodal&pin=40.717304%2C-**

73.956872&exclude_routes=%5B%22r-dr5r-l%22%5D) in Mapzen Mobility Explorer. Note that at the bottom of the left-hand column, you will see links to the API response for the Transitland and Mobility APIs that are used in Explorer queries. For the URLs that return GeoJSON, you can add them as a source in **Tangram Play (https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fmapzen.com%2Fapi%2Fscenes%2F22%2F427%2Fresources%2Fdefault.yaml#14.1374/40.7271/-73.9780)** and draw transit lines stations and the isochrones together.

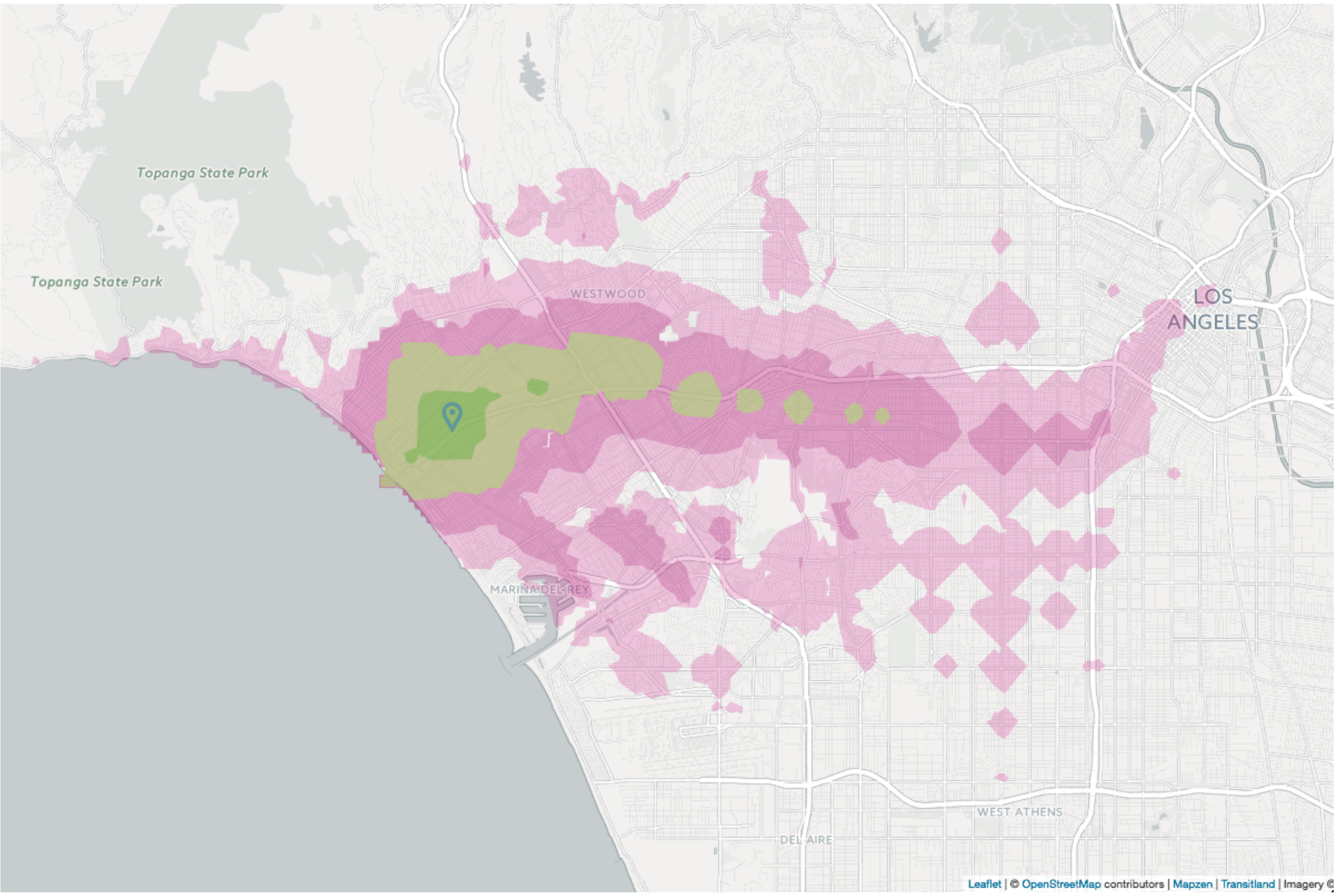
This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Exposing service improvements

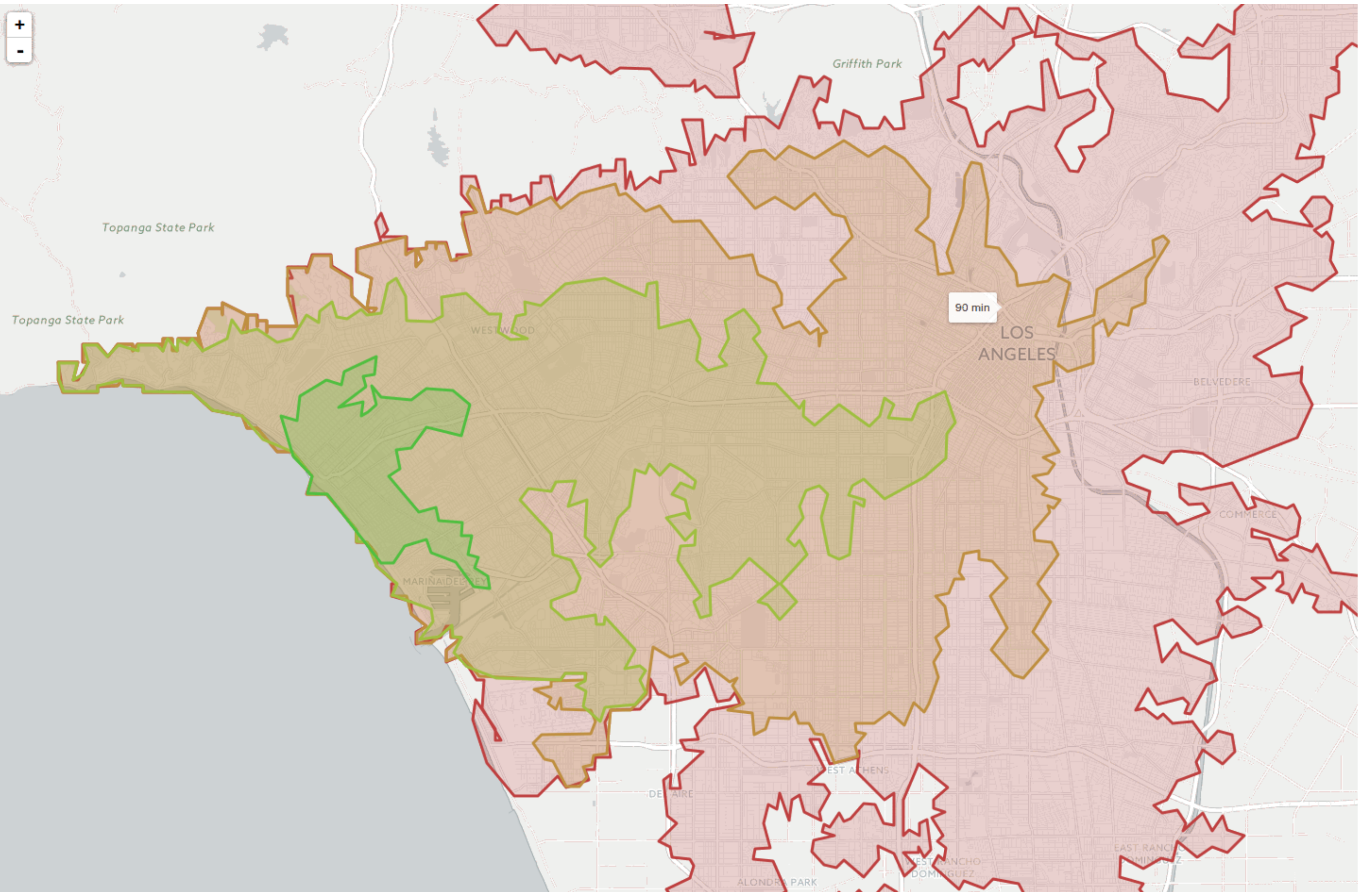
The `include` feature can be used to show how things have gotten better after transit service was added.

Los Angeles’ **Expo Line (https://en.wikipedia.org/wiki/Expo_Line_%28Los_Angeles_Metro%29)** came online in May of 2016, providing service from Downtown Los Angeles to Santa Monica. You can now get from Santa Monica to downtown Los Angeles in just over an hour.

If we use our filters to exclude the Expo Line (https://mapzen.com/mobility/explorer/#/isochrones?bbox=-118.53801727294922%2C33.92655407861729%2C-118.05633544921874%2C34.13511003175251&exclude_routes=%5B%22r-9q5c-metroexpoline806%22%5D&isochrone_mode=multimodal&pin=34.0099810521927%2C-118.49407196044922), we can see what the service was like before 2016.


Topanga State Park
Topanga State Park
WESTWOOD
LOS ANGELES
MARINA DEL REY
WEST ATHENS
DEL AIRE
Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery ©

Before the Expo line, it would take riders more than 2 hours to get east of downtown LA!


+
-
Griffith Park
Topanga State Park
Topanga State Park
WESTWOOD
90 min
LOS ANGELES
BELVEDERE
COMMERCE
MARINA DEL REY
WEST ATHENS
DEL AIRE
ALONDRA PARK
WEST RANCHO DOMINGUEZ
EAST RANCHO DOMINGUEZ

Note that this GIF shows isochrones in 30 minute contours, which is something that can be ***adjusted in the Isochrone API (https://mapzen.com/documentation/mobility/isochrone/api-reference/)****.*

Our exclusion filter for the transit options would be as follows:

```
 "costing_options":{"transit":{"filters":{"routes":{"ids":["r-9q5c-
metroexpoline806"],"action":"exclude"}}}}
```

Customizing your route

Inclusion and exclusion can be used to all sorts of transit assessments. Let's say you live in New Jersey and are heading to Manhattan to spend a day with the family. You only have a PATH SmartLink Card and don't feel like purchasing another fare card for a different transit operator. Valhalla will now allow you to only return transit routes and stops that have service from your favorite transit operator!

In this scenario, all you have to do is request a transit route that only includes PATH. Our origin is Newark Penn Station and our destination is West 30th Street @ 8th Ave. We are leaving on the last Tuesday in the month @ 8:00 AM. Typically, this transit route will take service provided by

Amtrak.

🚆	Enter the Newark Penn Station Amtrak Station.	227 m
🚆	*Depart: 8:12 AM from Newark Penn Station Amtrak.* Take the Keystone Service toward New York Penn Station. (1 stop) *Arrive: 8:30 AM at New York Penn Station.*	14.467 km
🚆	Exit the New York Penn Station.	52 m

But to keep that SmartLink card from burning a hole in your pocket, just reroute to only include/use the PATH operator.

🚆	Enter the Newark Station.	115 m
🚆	*Depart: 8:06 AM from Newark.* Take the PATH toward World Trade Center. (3 stops) *Arrive: 8:21 AM at Grove Street.*	11.009 km
🚆	*Depart: 8:22 AM from Grove Street.* Transfer to take the PATH toward 33rd Street. (6 stops) *Arrive: 8:40 AM at 33rd Street.*	6.021 km
🚆	Exit the 33rd Street Station.	1 m

How did we do it?

When a Valhalla tile is loaded into memory for a route, we populate three lists for including or excluding transit routes (stops, routes, and operators). Next, we added filters to the transit costing options so that a user can include or exclude as desired. Finally, when the multimodal algorithm is finding the best path, we determine if a stop, route, or operator should be excluded from the path. Everything, is filtered by Transitland's **Onestop IDs (https://transit.land/documentation/onestop-id-scheme/)** which are assigned during the transit import process.

How to obtain Onestop IDs for inclusion and exclusion?

- **Valhalla (https://mapzen.com/documentation/mobility/turn-by-turn/api-reference/)** - Onestop IDs can be obtained by running a transit route and parsing the returned JSON maneuver `transit_info` to obtain `operator_onestop_id` and the route `onestop_id` . A `transit_stop` contains the `onestop_id` for the stop.
- **Mobility Explorer (https://mapzen.com/mobility/explorer/)** - Onestop IDs can be obtained by selecting a single route, stop, or operator by clicking on it on the map. For routes and operators, you may also select by choosing from the drop-down menu in the sidebar. The Onestop ID will then be provided in the sidebar, along with other detailed information about the route, stop or operator you have chosen. More information can be found in the **Mobility Explorer documentation (https://mapzen.com/documentation/mobility/explorer/overview/)**.
- **Transitland (https://transit.land/)** - The Transitland Datastore API can be used to directly query for stops, routes, and operators using a number of options. For example, filtering for only **subway routes (http://transit.land/api/v1/routes?vehicle_type=metro)** or **bus routes (http://transit.land/api/v1/routes?vehicle_type=bus)**. Please see the **Transitland Datastore API documentation (https://transit.land/documentation/datastore/api-endpoints.html)** for more details.

New Filter Costing Options.

New Transit option	Description

New Transit option	Description

`filters`	Filters are used to include or exclude certain stops, routes, and operators. Filters must contain a list of Onestop IDs and an action.
	• `filters` (one or many can be used)
	  ◦ stops
	  ◦ routes
	  ◦ operators
	• `ids` - any number of Onestop IDs (e.g., o-9q9-bart)
	• `action`
	  ◦ exclude - exclude all of the ids listed in the filter
	  ◦ include - include only the ids listed in the filter
	Filters example - `"costing_options":{"transit":{"filters":{"stops":{"ids":["s-dr5rsq8pqg-8st~nyu<r21n","s-dr5rsr9wyg-14st~unionsq<r20n"],"action":"exclude"},"routes":{"ids":["r-dr5r-r"],"action":"exclude"},"operators":{"ids":["o-dr5r-path"],"action":"include"}}}}`

Using our example from New Jersey to Manhattan, our filter would look like this to only use PATH as our operator.

```
 "costing_options":{"transit":{"filters":{"operators":{"ids":["o-dr5r-path"],"action":"include"}}}}
```

Excluding & including in a city near you

Mapzen's motto is "start where you are", so if you see ways to improve our multimodal routing **please let us know (mailto:hello@mapzen.com?subject=)**. Also, have a look at our **documentation (https://mapzen.com/documentation/mobility/turn-by-turn/api-reference/)** to see the structure of the API request and response and **sign up for a free developer key to start using Mapzen Turn-by-Turn (https://mapzen.com/developers/sign_up)**.

Please help expand Transitland's coverage by **contributing feeds to the Feed Registry (https://transit.land/news/2016/02/19/get-started-add-feeds.html)**. In the coming months, we'll be ingesting even more of these feeds into Turn-by-Turn. To see which transit operators are routable, follow **this link to a filtered view of the Feed Registry (https://transit.land/feed-registry/?import_level=4)**. As always, should you have any questions, write to us at **hello@mapzen.com (mailto:hello@mapzen.com)**.

· 20 January 2017 ·

Greg Knisely

Greg is a data munger and software engineer for Mapzen's routing software.

Meghan Hade

Meghan is a software engineer with a background in urban planning on Mapzen's Mobility team. She works in javascript and plays in calligraphy, block printing, and finding ways to stash n+1 bikes in a San Francisco apartment.

© 2017 Mapzen

Mapzen on Tizen

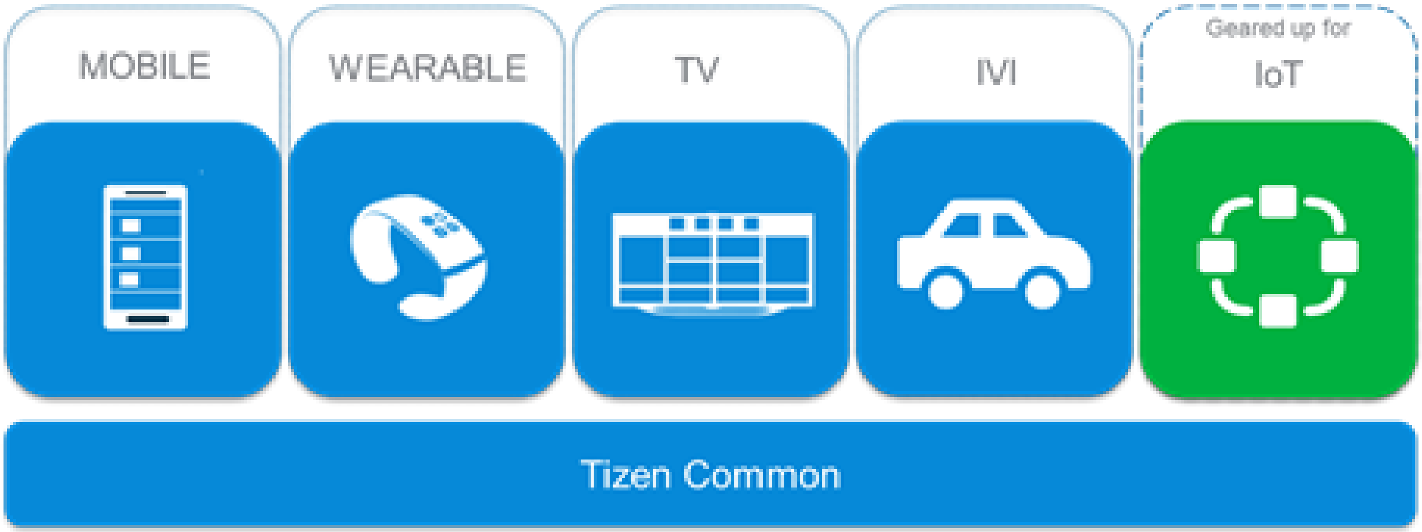
mobile (/tag/mobile) tangram (/tag/tangram)

Maps aren’t just for browsers and mobile anymore. Are you wearing a watch? Or looking at a TV? Now Mapzen can be there too — we’re happy to announce that the Tizen OS includes Mapzen as a map provider.

What is Tizen?

Tizen is an open-source operating system hosted by the **Linux Foundation (https://www.linuxfoundation.org/projects)**. Samsung Electronics is a significant contributor to Tizen, which it uses to power its smart TVs, smart watches, a **growing variety of IoT devices, and phones worldwide (https://wiki.tizen.org/wiki/Devices)**. As Samsung **moves to connect all its products to the internet (http://mashable.com/2015/01/05/samsung-internet-of-things/#QCu3q.nqSGqG)** we expect to see Tizen in more and more interesting devices.

Mapzen loves the open-source philosophy behind Tizen. Until now, HERE maps has been the only mapping option for developers in their applications. We’ve been working closely with the Tizen platform team at Samsung to bring our open mapping services to the core Tizen OS. With the launch of Tizen 3.0, Mapzen is now part of the core “**Tizen Common (https://source.tizen.org/release/tizen-common-milestones)**,” used to power all types of hardware.



How to get started

The Tizen version 3 API lets you **geocode, reverse geocode, calculate routes, search for places and view map widgets (https://developer.tizen.org/development/guides/native-application/location-and-sensors/maps-and-maps-service/)**. Just **sign up for a Mapzen API key (https://developer.tizen.org/development/guides/native-application/location-and-sensors/maps-and-maps-service/getting-mapzen-api-key)** and declare Mapzen as a map provider, and you're there.

We're excited to see you get Mapzen up on your wall, on your wrist, or even in your car! Let us know what you make! **No matter where you go, there you are (https://m.youtube.com/watch?v=jv_jkju_iZg)**.

· 24 January 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

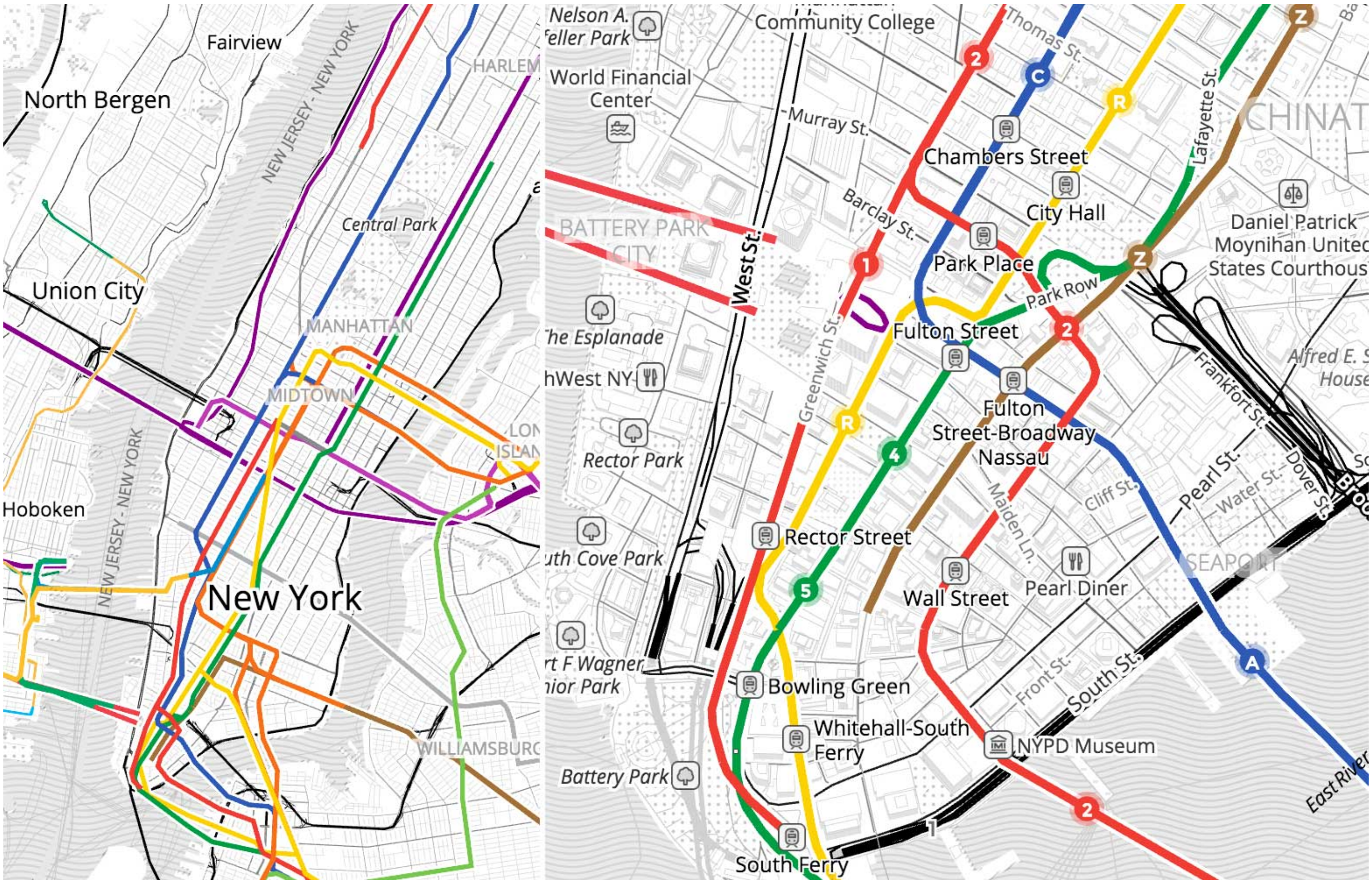
Mapzen Basemap Styles — SDK, label & API key updates

cartography (/tag/cartography) tangram (/tag/tangram)

We've made quite a few improvements to all of our Mapzen basemap styles, particularly with **place labels** and **SDK features** across the board.

This release also changes how **Mapzen API keys** are supported. This is a significant change. You will need to update how you import Mapzen basemap styles and set developer API keys within your map application. While this release is backwards compatible and makes setting a custom Mapzen API key optional, the next major release in March ***will*** require the custom key. Details on setting a Mapzen API key are available below.

All six Mapzen basemap styles, their three label variants, and dozens of SDK options give you powerful ways to personalize the map, add data visualization overlays, display search results, and route to your destination.

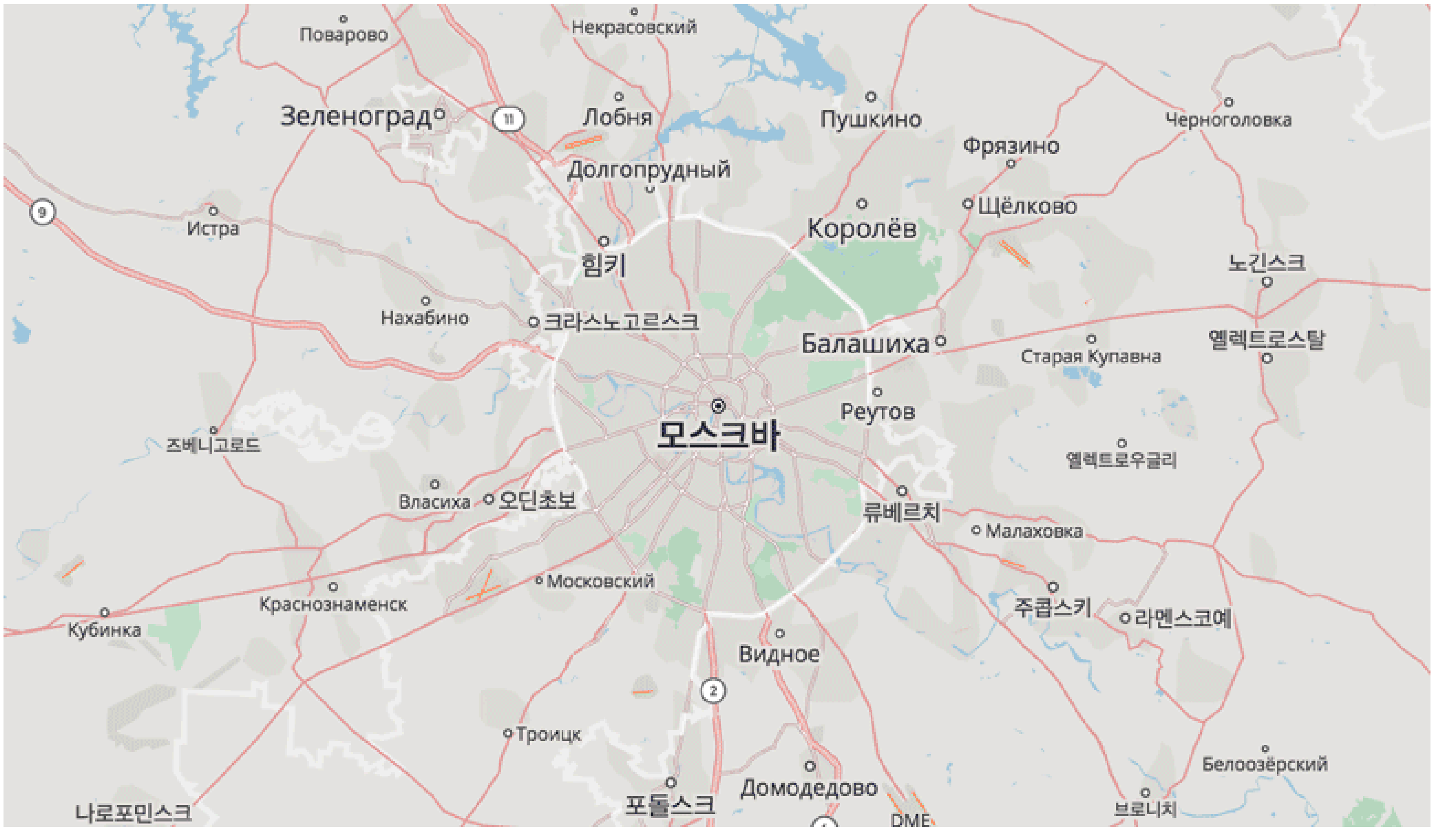

Fairview
North Bergen
NEW JERSEY - NEW YORK
HARLEM
Central Park
Union City
MANHATTAN
MIDTOWN
Hoboken
NEW JERSEY - NEW YORK
New York
WILLIAMSBURG
Nelson A.
Community College
Thomas St.
World Financial Center
Murray St.
CHINAT
Chambers Street
Lafayette St.
City Hall
Barclay St.
BATTERY PARK CITY
West St.
Daniel Patrick Moynihan United States Courthouse
Park Place
Park Row
The Esplanade
Fulton Street
Greenwich St.
Alfred E. S House
NY Waterway
Fulton Street-Broadway Nassau
Frankfort St.
Rector Park
Cliff St.
Pearl St.
Water St.
Dover St.
Maiden Ln.
Rector Street
Pearl Diner
SEAPORT
Wall Street
F Wagner Park
Bowling Green
Front St.
South St.
Whitehall-South Ferry
NYPD Museum
Battery Park
East River
South Ferry
1
2
C
R
Z
4
5
A

*Transit overlay SDK option added to Refill, with new line shields (**check them out full screen (https://mapzen.com/tangram/view/?scene=https://mapzen.com/api/scenes/22/164/resources/empty.yaml#14.7875/35.6791/139.7618)**)*

Mapzen basemap SDK changes:

- **Mapzen API key** support added to all styles
- **Language** support to Walkabout, Cinnabar, and Zinc
- **Transit overlay** support to Refill, Walkabout, Tron, Cinnabar, and Zinc
- **Transit user experience styles** for routing to Refill, Cinnabar, and Zinc
- **Data visualization** support to Cinnabar and Zinc
- **Labels** support for disabling road shields to all styles
- **Basemap styling** support for disabling building extrusion to all styles
- **Generic shields** available to all styles
- **Icon library** support to all styles

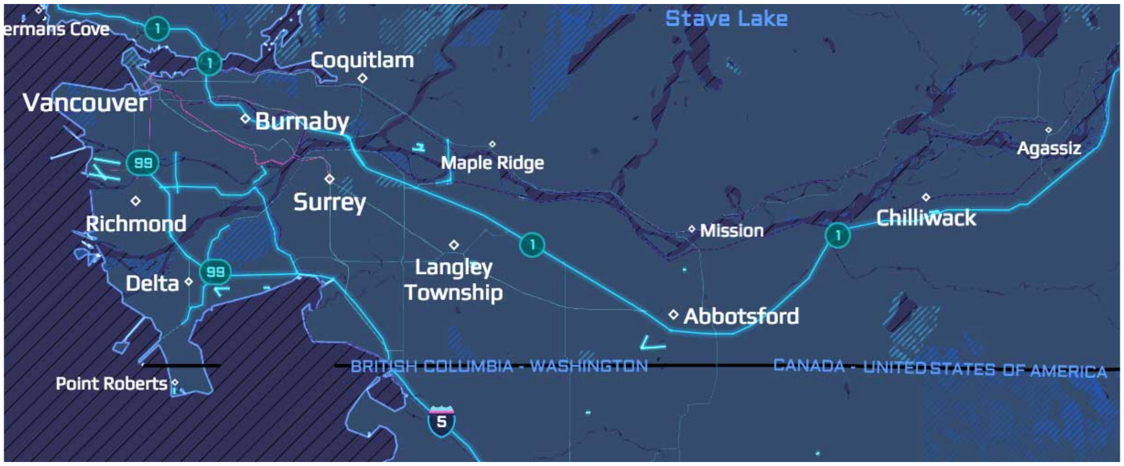
Cartography **documentation (https://mapzen.com/documentation/cartography/styles/)** has been updated, and the **SDK reference (https://mapzen.com/documentation/cartography/api-reference/)** includes usage examples.


Поварово
Некрасовский
Зеленоград
11
Лобня
Пушкино
Фрязино
Черноголовка
Долгопрудный
9
Истра
Щёлково
Королёв
힘키
노긴스크
Нахабино
크라스노고르스크
Балашиха
Старая Купавна
엘렉트로스탈
Реутов
즈베니고로드
모스크바
엘렉트로우글리
Власиха
오딘초보
류베르치
Малаховка
Московский
Краснознаменск
주콥스키
라멘스코예
Кубинка
Видное
2
Троицк
Белоозёрский
Домодедово
포돌스크
나로포민스크
DME
브로니치

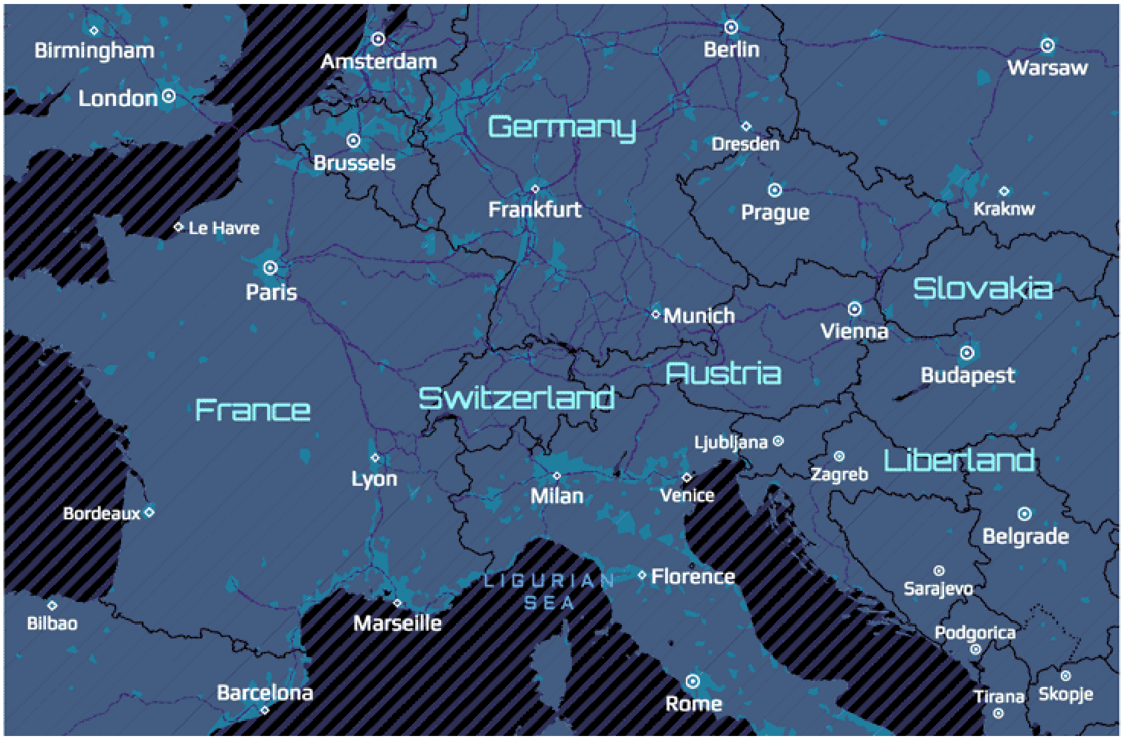
House styles in Moscow, Osaka, Kyoto, Cairo and Beijing in various languages. (above)

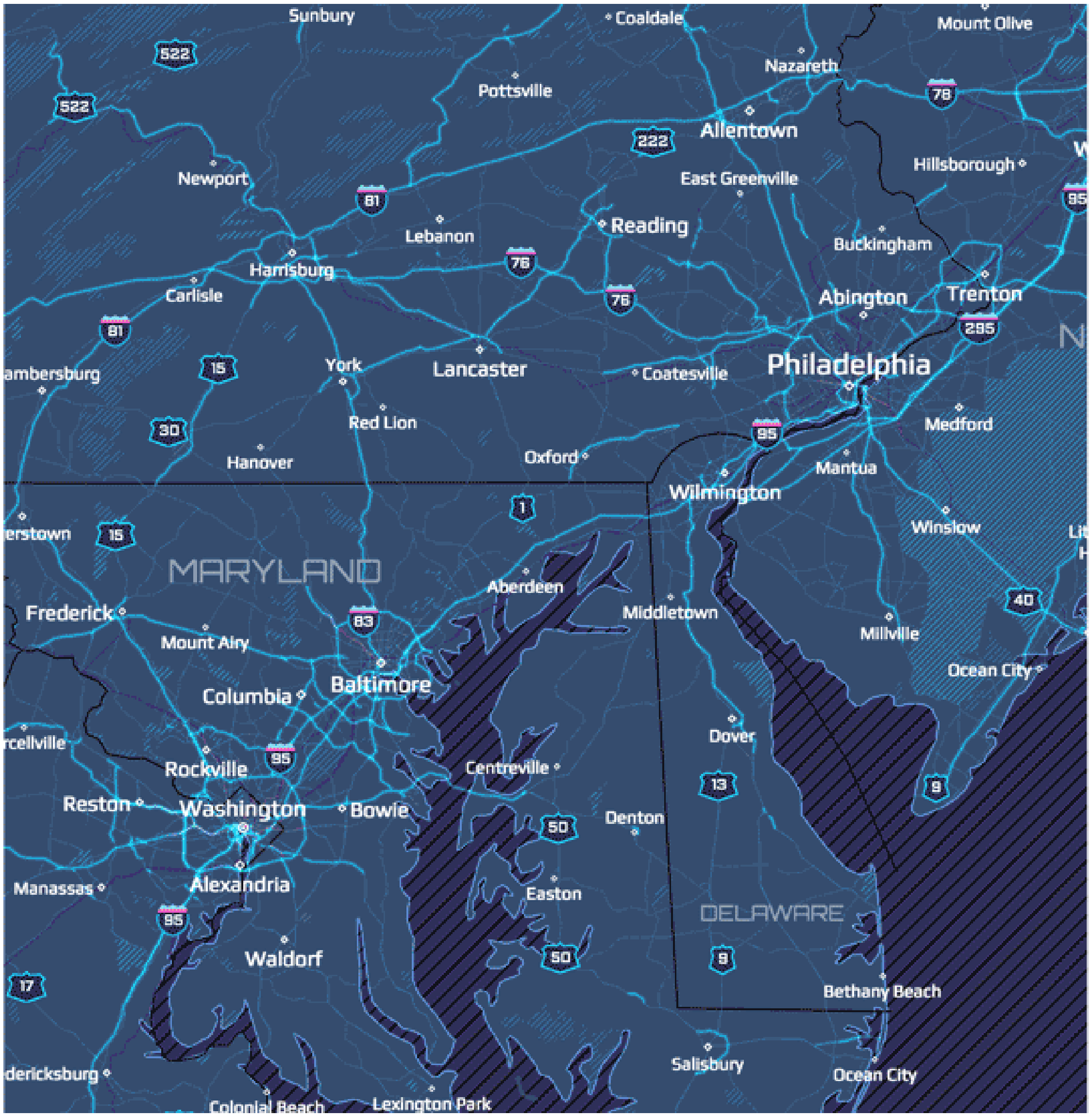
Label Improvements

- Behind the scenes we've improved **performance** across all house styles with refactored scene files. This has resulted in speedups of 10-30% at many zooms (especially with dense labels).
- Boundary line labels added

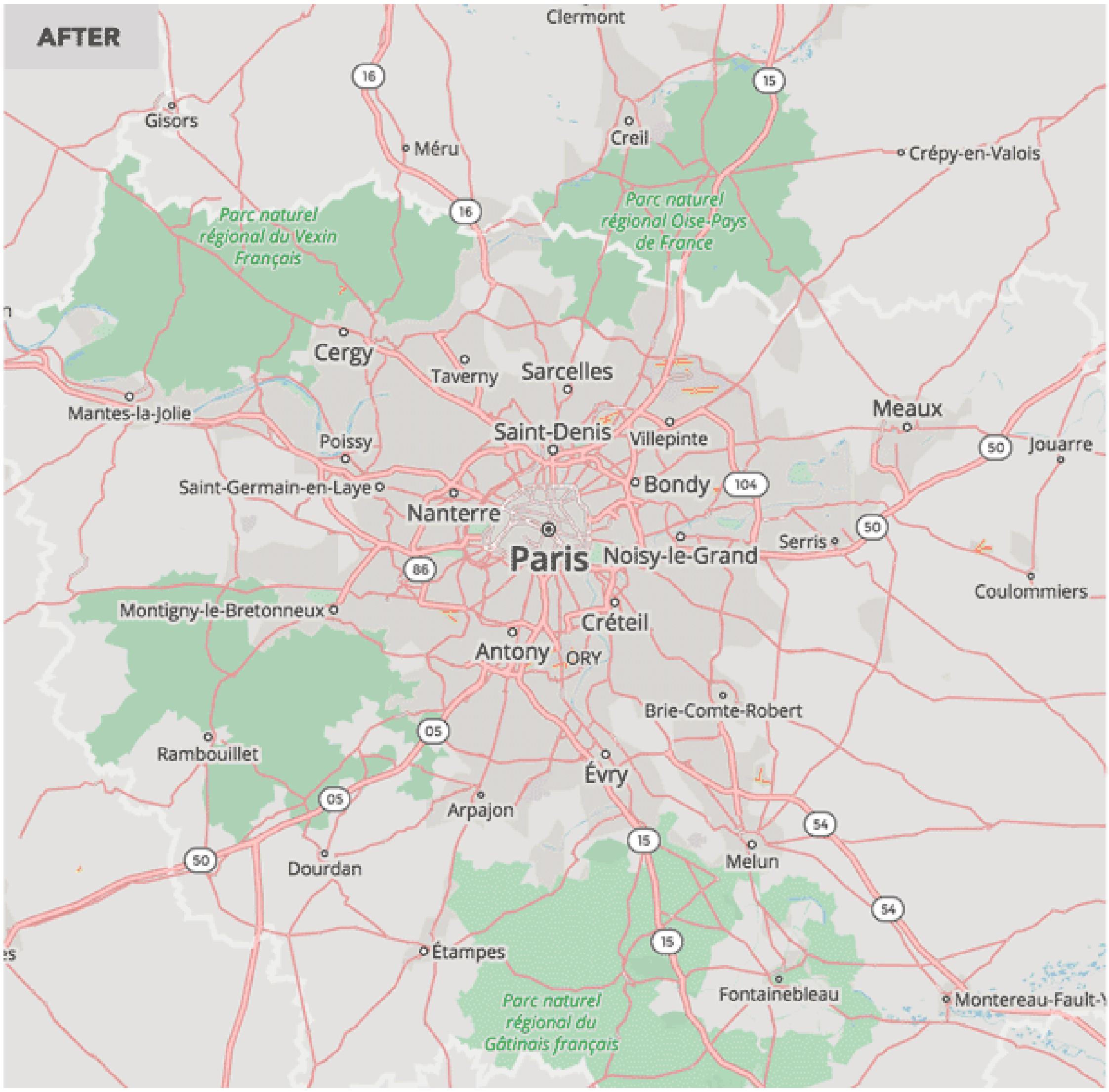

ermans Cove
Stave Lake
Coquitlam
Vancouver
Burnaby
Maple Ridge
Agassiz
Surrey
Richmond
Mission
Chilliwack
Langley Township
Delta
Abbotsford
Point Roberts
BRITISH COLUMBIA - WASHINGTON
CANADA - UNITED STATES OF AMERICA
1
99
5

- Low zoom labels are more consistent

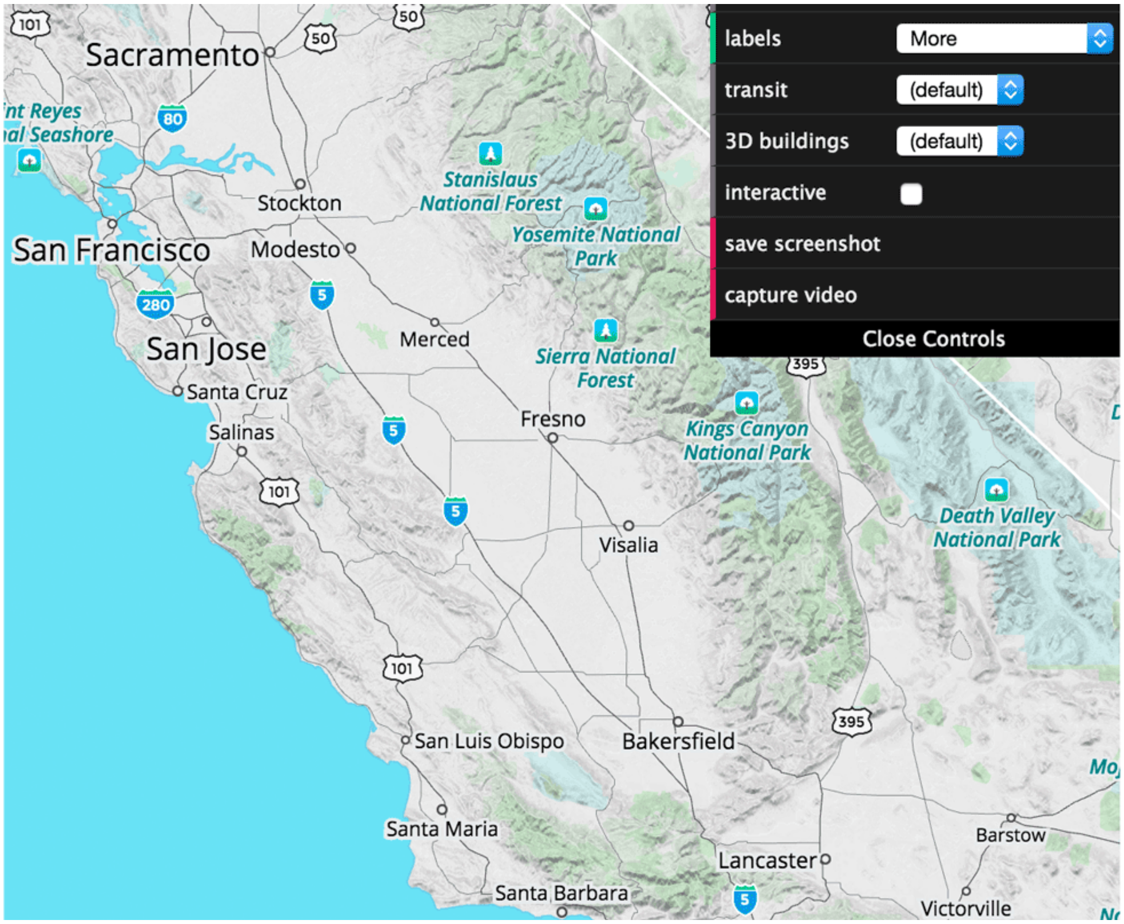

Birmingham
Amsterdam
Berlin
Warsaw
London
Germany
Dresden
Brussels
Frankfurt
Prague
Kraknw
Le Havre
Paris
Slovakia
Munich
Vienna
Budapest
Austria
France
Switzerland
Ljubljana
Zagreb
Liberland
Lyon
Milan
Venice
Bordeaux
Belgrade
LIGURIAN SEA
Florence
Sarajevo
Bilbao
Marseille
Podgorica
Barcelona
Rome
Tirana
Skopje


Sunbury
Coaldale
Mount Olive
522
Nazareth
522
Pottsville
78
222
Allentown
East Greenville
Hillsborough
Newport
81
Reading
Lebanon
Buckingham
78
Harrisburg
76
Carlisle
Abington
Trenton
81
295
Philadelphia
15
York
Lancaster
Coatesville
ambersburg
Red Lion
Medford
30
95
Hanover
Oxford
Mantua
Wilmington
1
erstown
15
Winslow
MARYLAND
Aberdeen
Middletown
40
Frederick
83
Millville
Mount Airy
Baltimore
Ocean City
Columbia
Dover
rcellville
Centreville
95
Rockville
13
9
Reston
Washington
Bowie
Denton
50
Manassas
Alexandria
Easton
95
DELAWARE
Waldorf
50
9
17
Bethany Beach
Salisbury
dericksburg
Ocean City
Colonial Beach
Lexington Park

- Improved label hierarchy with more locality labels and typographic improvements

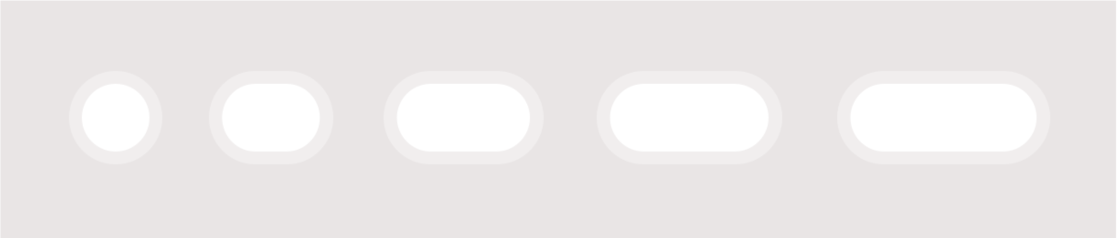

AFTER
Clermont
Gisors
Méru
Creil
Crépy-en-Valois
Parc naturel régional du Vexin Français
Parc naturel régional Oise-Pays de France
Cergy
Taverny
Sarcelles
Mantes-la-Jolie
Meaux
Saint-Denis
Villepinte
Poissy
Jouarre
Saint-Germain-en-Laye
Bondy
Nanterre
Serris
Paris
Noisy-le-Grand
Coulommiers
Montigny-le-Bretonneux
Créteil
Antony
ORY
Brie-Comte-Robert
Rambouillet
Évry
Arpajon
Dourdan
Melun
Étampes
Fontainebleau
Montereau-Fault-
Parc naturel régional du Gâtinais français
16
15
50
104
86
05
54

- Label variants consistent across all styles (option to choose *more labels*, *some labels*, *some labels/no shields*, *none*). For example in Walkabout (below), the default *some labels* now have no icons, *more labels* includes icons, and includes the option to remove road shields.


Sacramento
Stockton
San Francisco
Modesto
San Jose
Santa Cruz
Salinas
Merced
Fresno
Visalia
Stanislaus National Forest
Yosemite National Park
Sierra National Forest
Kings Canyon National Park
Death Valley National Park
San Luis Obispo
Bakersfield
Santa Maria
Santa Barbara
Lancaster
Barstow
Victorville
101
50
80
280
5
395

labels: More
transit: (default)
3D buildings: (default)
interactive: ☐
save screenshot
capture video
Close Controls

- Generic shields available for use (similar to shields used in transit overlay)



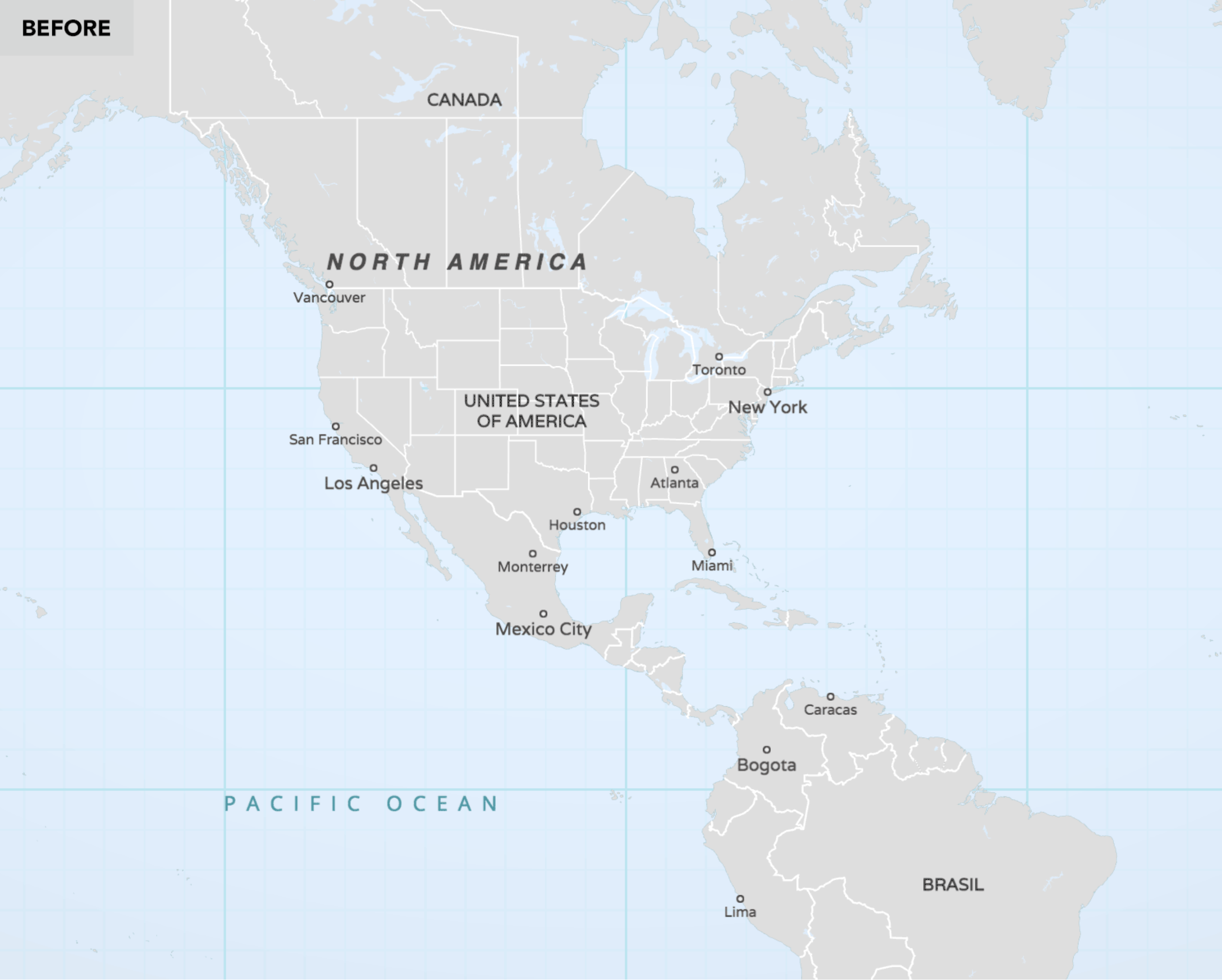
- Consistent compatibility with icon library (sprite names for points of interest)

Bubble Wrap Improvements

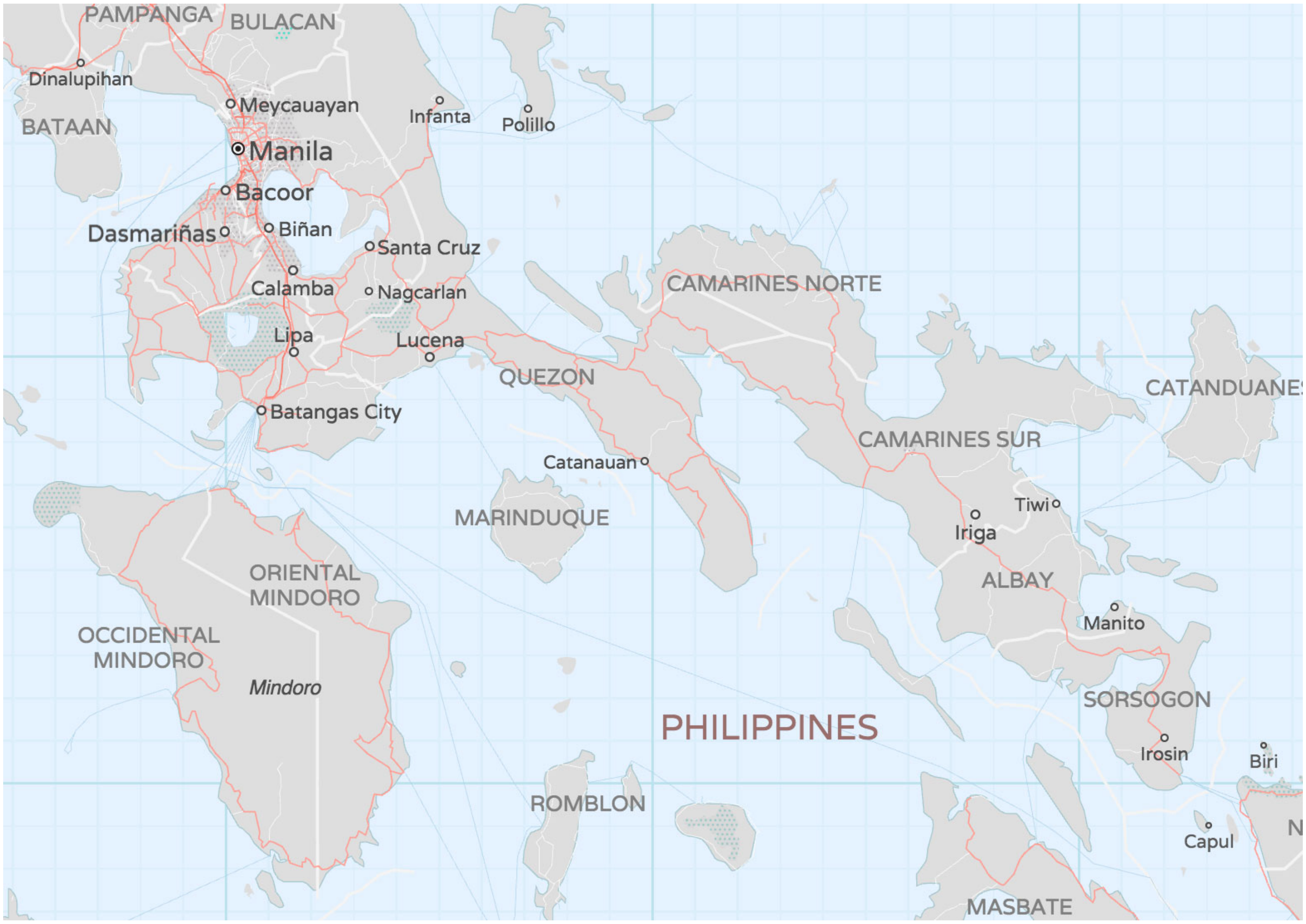
When we **launched Bubble Wrap (https://mapzen.com/blog/bubble-wrap-carto/)** a year ago it was a “mobile first” style showcased in the **Eraser Map (https://mapzen.com/blog/erasermap-beta/)** Android app. It was available in just a single variant with all points of interests labeled. Today we’re extending Bubble Wrap to match the general “some” (default), “more”, and “no” label pattern available in the other Mapzen basemap styles.

If you like the earlier Bubble Wrap with all the icons, migrate to using the new `bubble-wrap-more-labels` variant as icons and most labels are dropped from the default `bubble-wrap` variant.

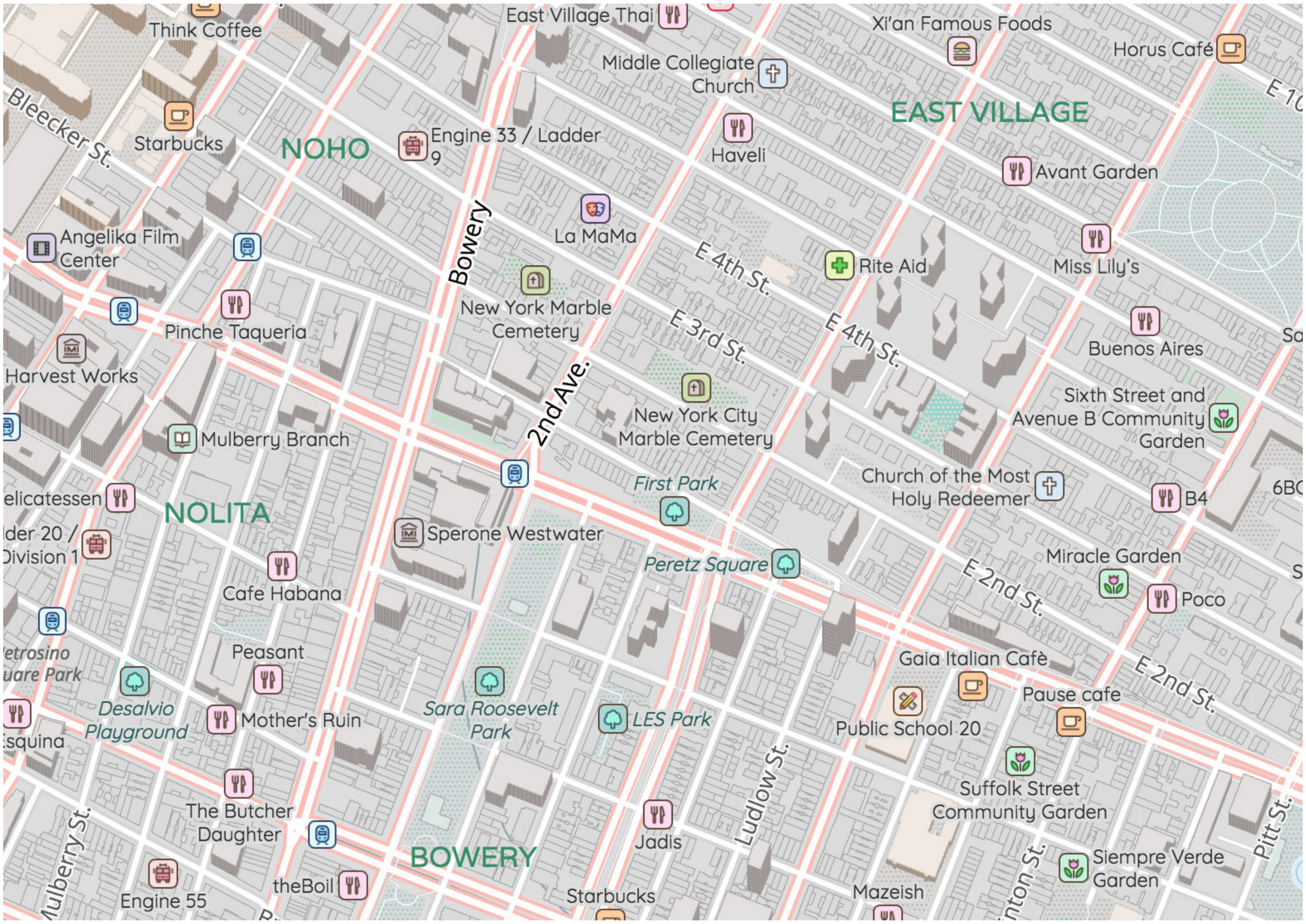
We've also made some new typographic improvements to place labels across all zoom levels from continents to neighborhoods.


BEFORE
CANADA
NORTH AMERICA
Vancouver
Toronto
UNITED STATES OF AMERICA
New York
San Francisco
Los Angeles
Atlanta
Houston
Monterrey
Miami
Mexico City
Caracas
Bogota
PACIFIC OCEAN
BRASIL
Lima

Bubble Wrap Low Zoom View of North America Labels (above)


PAMPANGA
BULACAN
Dinalupihan
Meycauayan
Infanta
Polillo
BATAAN
Manila
Bacoor
Dasmariñas
Biñan
Santa Cruz
CAMARINES NORTE
Calamba
Nagcarlan
Lipa
Lucena
QUEZON
Batangas City
CATANDUANE
CAMARINES SUR
Catanauan
MARINDUQUE
Tiwi
Iriga
ORIENTAL
MINDORO
ALBAY
OCCIDENTAL
MINDORO
Manito
Mindoro
SORSOGON
PHILIPPINES
Irosin
Biri
ROMBLON
Capul
N
MASBATE

Bubble Wrap View of Philippine Province Labels (above)


Think Coffee
East Village Thai
Xi'an Famous Foods
Horus Café
Middle Collegiate Church
Bleecker St.
Starbucks
NOHO
Engine 33 / Ladder 9
Haveli
EAST VILLAGE
Avant Garden
Bowery
La MaMa
Angelika Film Center
E 4th St.
Rite Aid
Miss Lily's
New York Marble Cemetery
Pinche Taqueria
E 3rd St.
E 4th St.
Buenos Aires
Harvest Works
2nd Ave.
New York City Marble Cemetery
Sixth Street and Avenue B Community Garden
Mulberry Branch
First Park
Church of the Most Holy Redeemer
B4
NOLITA
Sperone Westwater
Peretz Square
Miracle Garden
Cafe Habana
E 2nd St.
Poco
Peasant
Gaia Italian Cafè
Desalvio Playground
Mother's Ruin
Sara Roosevelt Park
LES Park
Pause cafe
E 2nd St.
Public School 20
Suffolk Street Community Garden
The Butcher Daughter
Jadis
Ludlow St.
Mulberry St.
BOWERY
Siempre Verde Garden
Pitt St.
theBoil
Engine 55
Starbucks
Mazeish

Bubble Wrap View New York Downtown Neighborhood Labels (above)

Introducing more ZIP bundles

As we've grown the number of fonts and image assets required for Mapzen basemap styles, it's gotten a little confusing about which assets are needed to load a style. When we published Tron we introduced the notion of `.zip` bundles for styles. We've extended bundle support to all Mapzen basemap styles (yet retain the earlier `.yaml` support).

Mapzen.js been updated with the new resources and the source files are available on the Mapzen CDN for direct Tangram YAML `import` — see Cartography **documentation (https://mapzen.com/documentation/cartography/styles/)** for URLs. We recommend pegging imports to a **major version** to reduce risk that future Mapzen basemap updates become incompatible with your custom map content and styling (e.g. `https://mapzen.com/carto/refill-style/6/refill-style.zip` as opposed to `https://mapzen.com/carto/refill-style/refill-style.zip`).

Need to bundle your own Tangram scene assets? There's a **tool (https://github.com/tangrams/bundler)** for that.

Required soon: a Mapzen API key

This is the final major version of Mapzen basemap styles that will not require a developer API key. We've seen a dramatic increase in usage of Mapzen's shared services — signing up for a developer API key will continue your existing free access to the shared services, and will allow Mapzen to guard against abuse. *Please update your maps with a Mapzen API key before March.*

While earlier versions of the basemap styles shipped with an embedded API key to access Mapzen vector and terrain tiles, the new versions enable you to specify your own free Mapzen developer API key. This is done using a new Tangram scene file global variable provided in all Mapzen basemap styles.

Starting in March 2017, maps that do not set an developer API key will no longer function, so please go **sign up (https://mapzen.com/documentation/overview/)**.

Example **Tangram YAML** usage:

```
import: https://mapzen.com/carto/refill-style/6/refill-style.yaml

global:
  sdk_mapzen_api_key: mapzen-xxxxxx
```

Example **Tangram JS** usage:

```
var layer = Tangram.leafletLayer({
  scene: {
    import: 'https://mapzen.com/carto/refill-style/6/refill-style.yaml',
    global: { sdk_mapzen_api_key: 'mapzen-xxxxxx' }
  });
```

Example **Mapzen.js** usage:

```
<script>
  L.Mapzen.apiKey = 'mapzen-xxxxxx';

  var map = L.Mapzen.map('map', {
    center: [40.8041, -124.1506],
    zoom: 15,
    maxZoom: 20,
    tangramOptions: {
      scene: L.Mapzen.BasemapStyles.Refill
    }
  });
</script>
```

Explore all Mapzen basemap styles below:

Hoboken
New York
+
−

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

Bubble Wrap

(These maps are interactive! Open full screen ➶ (https://mapzen.com/products/maps/) *)*

Note: This post was updated on May 22, 2017 to reflect new documentation URLs and rate limit information.

· 27 January 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

Mapzen bans Trump for terms of service violation

We've been fortunate lately to have many news outlets publishing maps with Mapzen tools and services. **The most recent was the Washington Post (https://www.washingtonpost.com/news/politics/wp/2017/01/27/the-areas-where-trumps-wall-would-be-built-voted-heavily-against-trump/?utm_term=.6c709c9efd1c)**, with a dynamic map exploring Trump's proposed border wall.

!

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

(Map by Philip Bump, ***Washington Post (https://www.washingtonpost.com/news/politics/wp/2017/01/27/the-areas-where-trumps-wall-would-be-built-voted-heavily-against-trump/?utm_term=.6c709c9efd1c)****. Congressional vote*

*data from **DailyKos (http://dailykos.com/story/2017/1/3/1616470/-Check-out-our-comprehensive-115th-Congress-guide-with-election-data-demographics-and-member-stats)**. Wall data by Michael Corey, **Reveal News (https://www.revealnews.org/author/michaelcorey/)**.)*

An interesting question came up this weekend on social media: what if Trump uses Mapzen tools to construct the Mexican border wall? How would we respond? Large-scale mapping services would be necessary for a project like this, and many mapping companies focus on the government as an important source of revenue. **ESRI (http://www.esri.com/industries/federal)**, for example, provides mapping services to a wide range of government clients.

If you check Mapzen's **developer terms of service (https://mapzen.com/terms/)**, we prohibit "promotions of hate or incitement of violence." After his first week in office, we believe Trump's administration to be in violation of this critical section of our terms, and we would actively block any use of our services by relevant departments of the executive branch. In our view, the current President promotes hate and violence and we refuse to support it. We are proactively banning Trump for terms of service violation, for 120 days (at least). We encourage other mapping providers to take a similar stance and review their government contracts with a critical eye.

Otherwise, we welcome and encourage anyone to use our tools to understand and explain our complex world!

· 29 January 2017 ·

Randy Meech

Billerica Memorial High School graduate. BA, MTS. Mapzen CEO 2013-present.

© 2017 Mapzen

Coloring in your commute

tangram (/tag/tangram) cartography (/tag/cartography)

We’ve enabled a transit layer across all our **updated basemap styles (https://mapzen.com/blog/basemap-style-update/)**! It pulls **colors and information needed to draw subways, trains, and their labels (https://mapzen.com/blog/targeted-editing-transit-colours/)** from OpenStreetMap.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

***Zoom into Barcelona full screen (https://tangrams.github.io/tangram-frame/?scene=https://mapzen.com/api/scenes/22/173/resources/scene.yaml#13/41.3939/2.1696)**!*

(Rail lines appear at zoom level 5, subway lines at z8, train stations at z10, subway stations at z12 and subway line labels at z13.)

Just like **enabling languages (https://mapzen.com/blog/languages-of-india/)**, it's very easy to turn on the transit layer after importing a basemap style in Tangram – just import the basemap and add two lines!

```
import: https://mapzen.com/carto/refill-style/6/refill-style.yaml

global:
  sdk_transit_overlay: true
```

If you're using **mapzen.js (https://github.com/mapzen/mapzen.js)**:

```
var map = L.Mapzen.map('map', {
  center: [40.8041, -124.1506],
  zoom: 15,
  maxZoom: 20,
  tangramOptions: {
    scene: {
      import: L.Mapzen.BasemapStyles.Refill,
      global: {
        sdk_transit_overlay: true
      }
    }
  }
});
```

Give it a try – it will probably take you **less than a minute (https://mapzen.com/blog/one-minute-map/)**!

· 30 January 2017 ·

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Quality Issues

mobility (/tag/mobility) transitland (/tag/transitland)

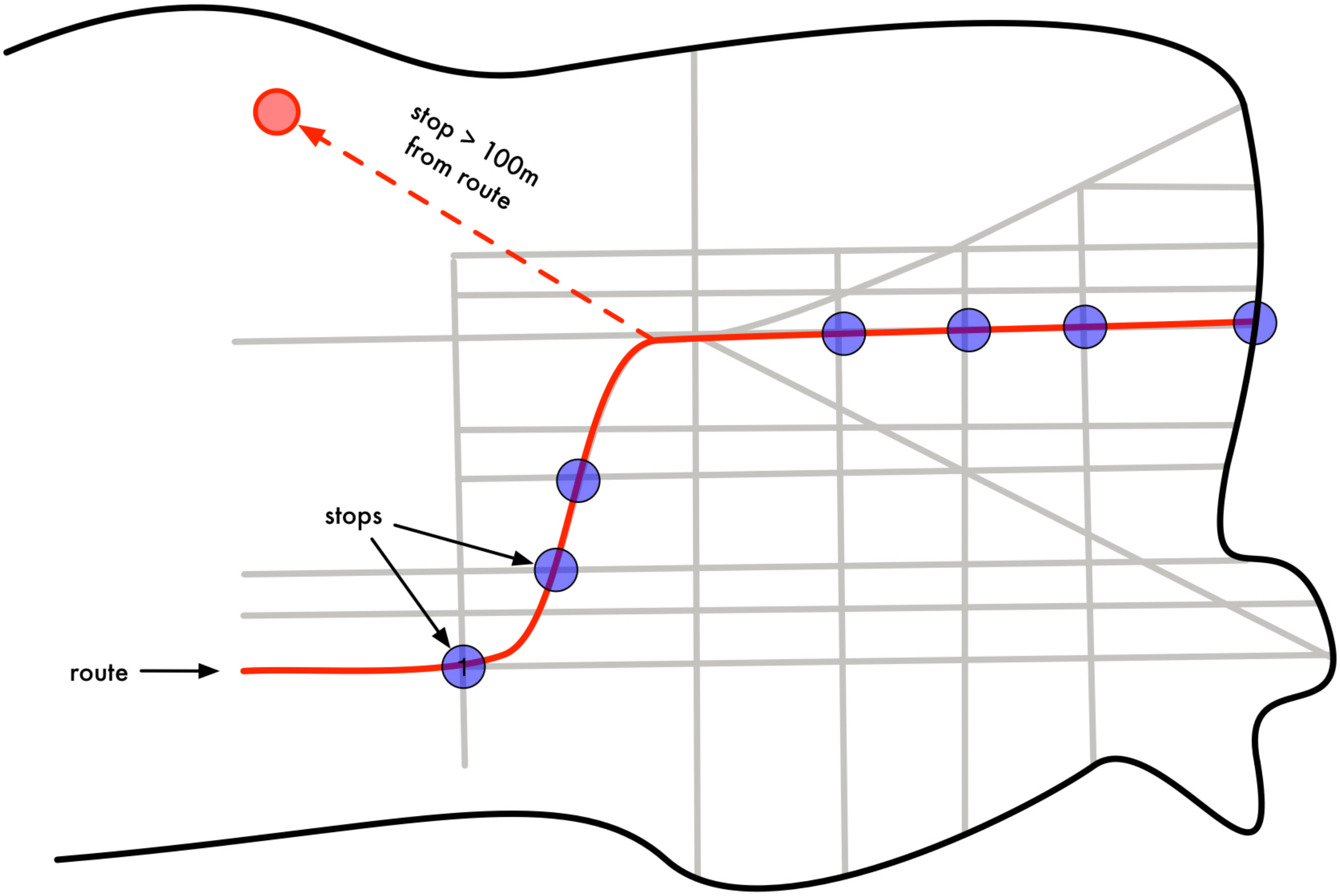
Transitland (https://transit.land) has issues. No, we're not talking about bugs, missing features, or flaws (we hope!) but a whole new way of tracking, viewing, and even a little editing of the qualitative concerns we find in the transit data we receive. We call these "Quality Issues."

Transit data is inherently tricky given all the spatial, temporal, and operational components that have to come together. Whether they are assembled and edited manually, or produced as the combined output of several different programs and processes, sometimes mistakes are made. Just to give a few examples, we've seen outdated and invalid URLs, stops that are too far from their routes, and trip shapes that don't match the trip's stop patterns or even go in the wrong direction.

It turns out that these errors are fairly easy to evaluate, maintain, and edit, especially when you see as many transit feeds as we do. While the transit data may be complex, it is also predictable, and we can build processes to detect common errors. Here's an example that illustrates our approach.

Suppose Transitland receives a feed version from the Fixme Transit Agency. During the import process, after the raw GTFS data has been consumed and translated into the Datastore's model, and before this data makes it into the database, all route stop patterns and their accompanying stops are quality checked for any unusually large distance gaps. In addition to being visually and factually inaccurate, these gaps can lead to mistakes in **how a routing engine computes a path (https://mapzen.com/blog/transitland-route-geometries/)** along the route line between the gapped stop and any other stop.

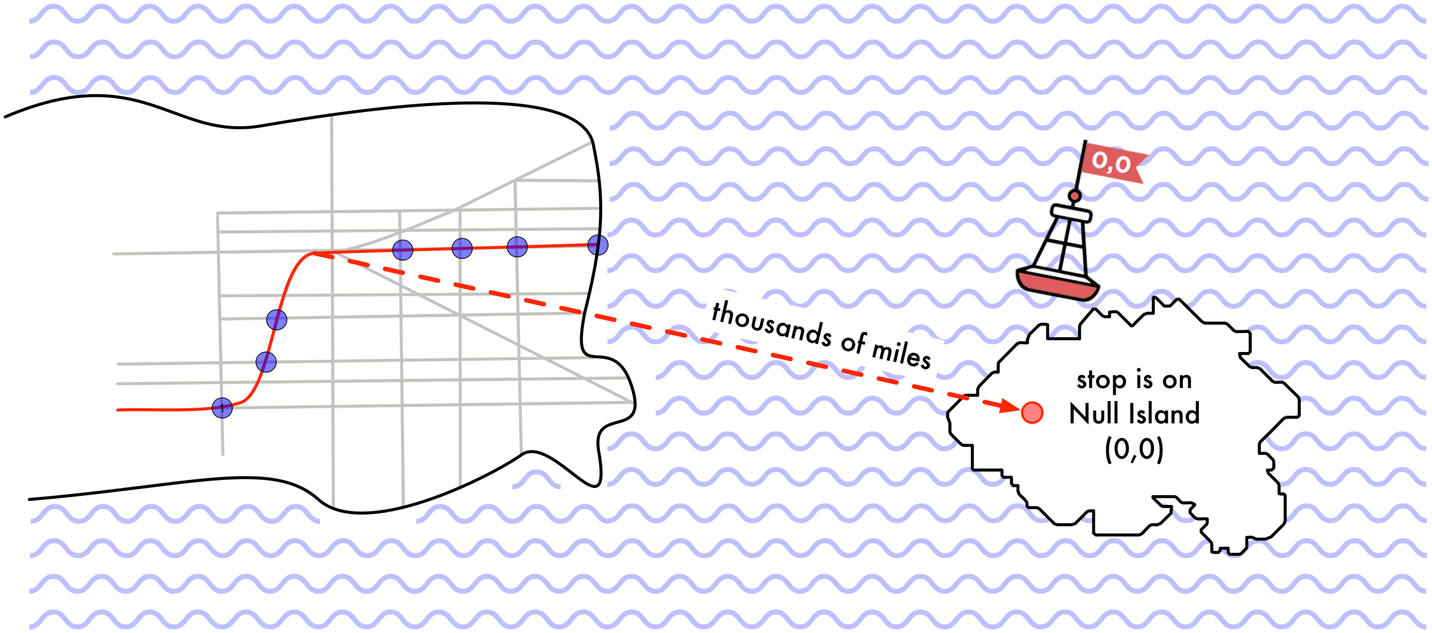
Everything seems good, except for one route stop pattern and stop pair in the "No Outlier Express" route. Due to a glitch in Fixme's scheduling software, stop "Main Street & 1st" is now located erroneously off of the route line in another part of town.



The distance gap function in the quality check process recognizes that this stop is unlikely to be so far away. It creates an issue record with an id “1” and reference to the specific route stop pattern and stop, and the attribute to “blame” for the issue. (In this case, both the route stop pattern line geometry and stop point geometry attributes are flagged as the “objects of blame”, because we can’t yet determine programatically which geometry is erroneous.)

A few weeks later, a new feed version is fetched by Transitland. This poses a problem for our Quality Issues process in that new data could make issues, like our issue “1”, irrelevant. To see this, suppose the import process finds and conducts a quality check on the previous route stop pattern and stop pair.

A different distance gap is found, but this time “Main Street & 1st” is now on **Null Island (https://blogs.loc.gov/maps/2016/04/the-geographical-oddity-of-null-island/)**. (Fixme has no ferry services to the middle of the Atlantic).

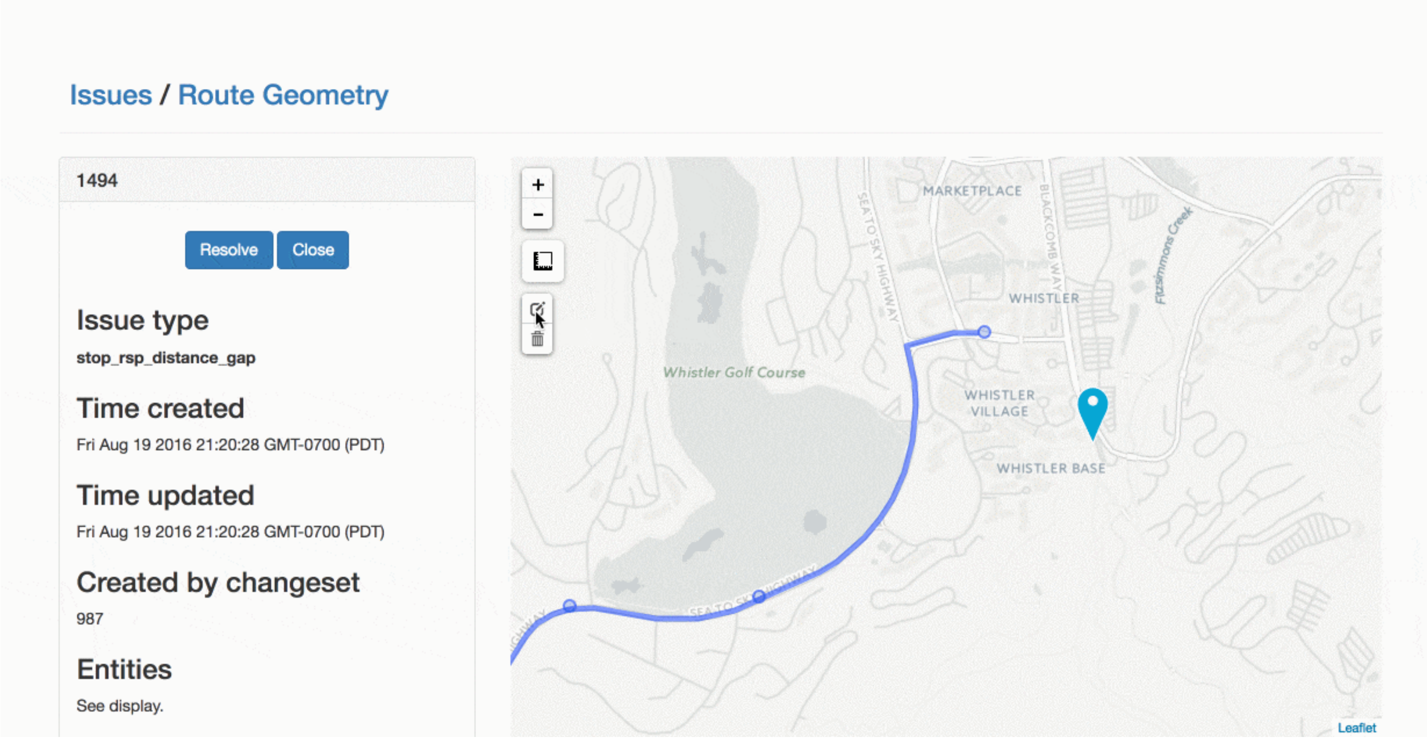


The details of the problem have changed, but the nature has not. So the previous quality issue “1” is deleted and a new issue record with id “2” is created. This new record is still almost identical to its predecessor, except for the id and a reference to the new feed version. (Note that in our current implementation, a new issue record will be created even if the coordinates of “Main Street & 1st” were still 0°N, 0°E in subsequent imports.)

Meanwhile, confused passengers, using an app based on Transitland’s data, have been queueing up in sailboats near Null Island only to miss their ride to work. To resolve this problem, someone submits a changeset with the field “issuesResolved” populated by the integer “2”. The changeset consists of a single payload: the “Main Street & 1st” onestop id and the geometry coordinates of that street corner. The changeset is applied, and once again quality checks are run. The Datastore verifies that the “issuesResolved” payload did not create an equivalent issue through quality checks (otherwise an error would be returned and the change would not be applied), and completes by writing the update to the database. Quality issue “2” is marked as closed, logged and deleted. During the next feed version import, the “Main Street & 1st” stop geometry is left unchanged from the edit, and no new issues are created.

A check on the distance gap between a stop and a route stop pattern is just one of a few quality checks conducted during changeset applications. Since there’s always more going on behind the scenes, please check out the **quality issues documentation (https://transit.land/documentation/datastore/quality_issues.html)** for more details.

Finally, we are working on some editing tools to help facilitate the resolution of issues with data edits. Stay tuned as we get them ready to release them to a wider audience!


Resolve
Close
+
-
MARKETPLACE
SEA TO SKY HIGHWAY
BLACKCOMB WAY
Fitzsimmons Creek
WHISTLER
Whistler Golf Course
WHISTLER VILLAGE
WHISTLER BASE
SEA TO SKY HIGHWAY
Leaflet

Issues / Route Geometry

1494

Issue type

stop_rsp_distance_gap

Time created

Fri Aug 19 2016 21:20:28 GMT-0700 (PDT)

Time updated

Fri Aug 19 2016 21:20:28 GMT-0700 (PDT)

Created by changeset

987

Entities

See display.

· 03 February 2017 ·

Alex Smith

© 2017 Mapzen

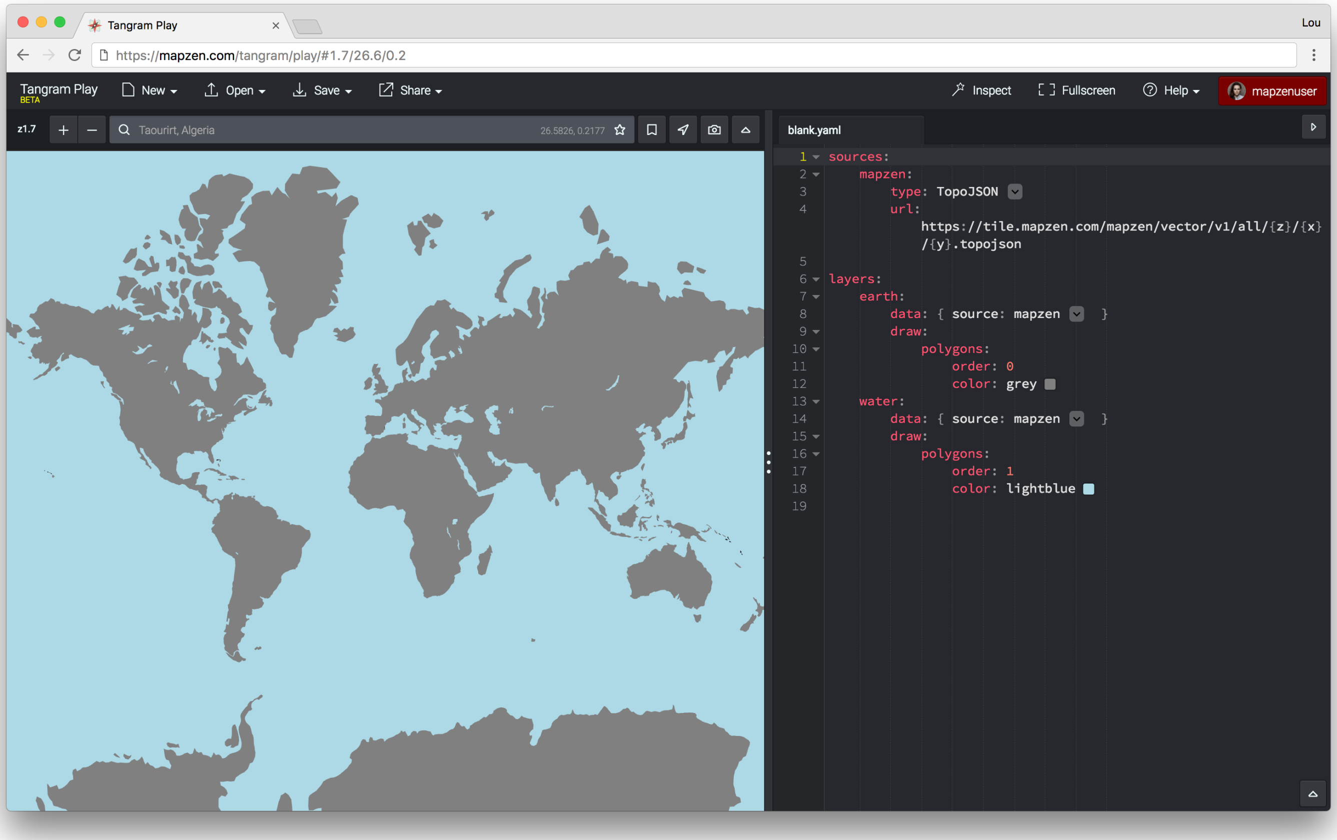
Tangram Play, now in public beta!

tangram (/tag/tangram)

Did you know that behind the scenes, we've been working on a code editor for Tangram scenes? We call it **Tangram Play (https://mapzen.com/tangram/play/)**. It's a browser-based, real-time editing tool for Tangram scene files. (If you've been following along with our **Make Your Own (https://mapzen.com/tag/make-your-own/)** series, you might have seen us link to it.)

As with all of our projects, we do all of our work **in the open (https://mapzen.com/blog/our-magna-carto/)**, so it's no surprise that members of our community have noticed it. Early adopters have already started using it — check out **Hacker Experience 2 (https://www.indiegogo.com/projects/hacker-experience-2-open-source-hacking-game-app)**, an indie game that uses Tangram Play to create its basemaps. Some **recent work with journalists (https://mapzen.com/blog/made-with-mapzen/)** was made with the help of Tangram Play (including this nice shoutout from **The Guardian (https://www.theguardian.com/cities/2016/sep/02/this-interactive-map-lets-you-watch-who-commutes-to-your-city-every-day)**!)

It's now time to bring this tool out from the early-adopter and internal-use only phase and make it more accessible for an even larger community of map designers and cartographers. So, today, we're proud to announce the public beta launch for **Tangram Play (https://mapzen.com/tangram/play/)**!


Tangram Play
Lou
https://mapzen.com/tangram/play/#1.7/26.6/0.2
Tangram Play BETA
New
Open
Save
Share
Inspect
Fullscreen
Help
mapzenuser
z1.7
Taourirt, Algeria
26.5826, 0.2177
blank.yaml

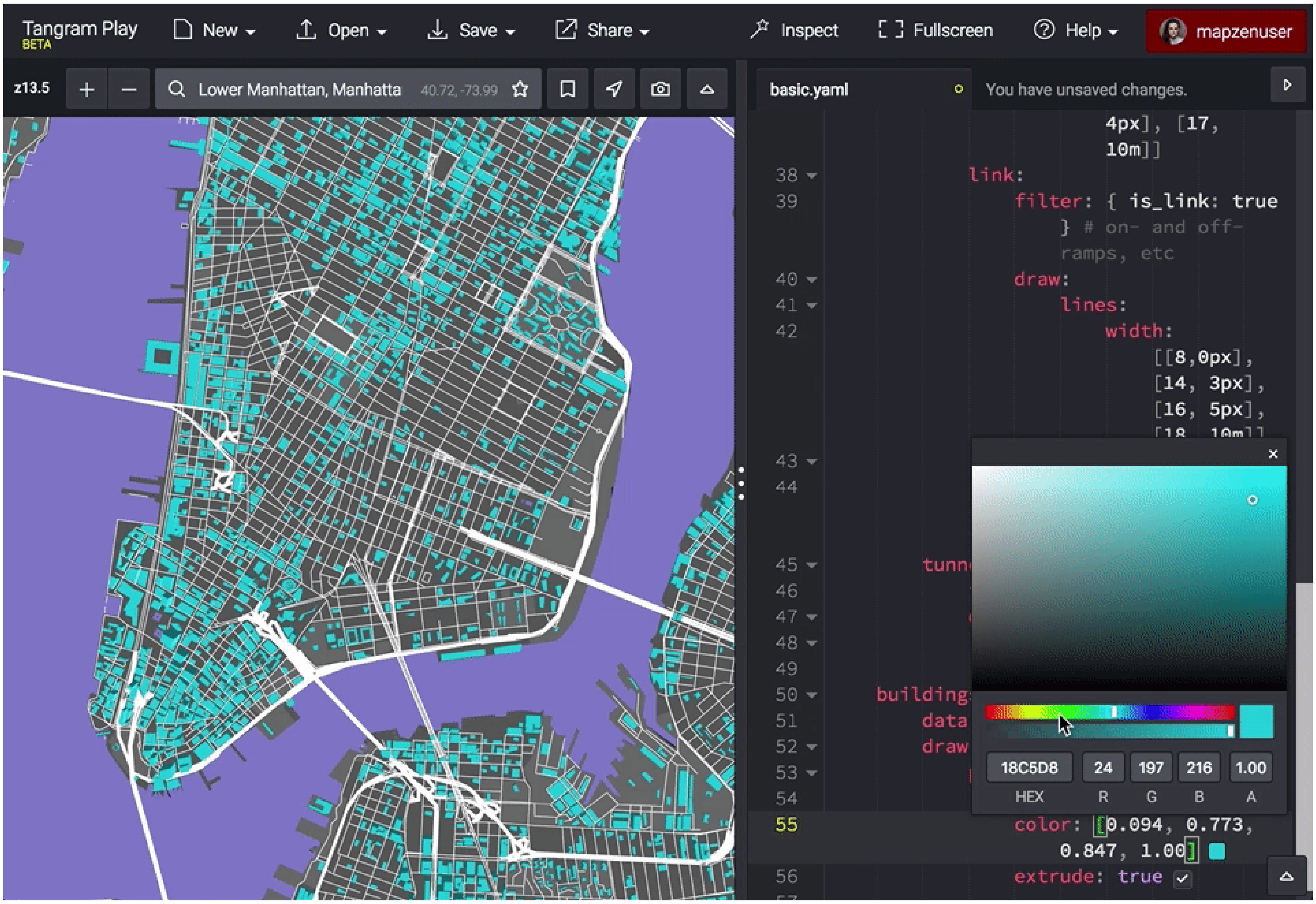
```
sources:
    mapzen:
        type: TopoJSON
        url:
            https://tile.mapzen.com/mapzen/vector/v1/all/{z}/{x}/{y}.topojson

layers:
    earth:
        data: { source: mapzen }
        draw:
            polygons:
                order: 0
                color: grey
    water:
        data: { source: mapzen }
        draw:
            polygons:
                order: 1
                color: lightblue
```

Playing with Play

You can start with a basic scene or open up one of **Mapzen's basemaps (https://mapzen.com/products/maps/)** to see the inner workings of a scene file in Tangram.

Some noteworthy features -

- Helpers like color pickers to help select values for different Tangram properties
- Easily move around to different locations in the world and bookmark **important locations (http://mike.teczno.com/notes/lake-merritt.html)** to make sure your designs hold up in different parts of the world and at varying zoom levels. You can also take a screenshot or record animations of the scene.

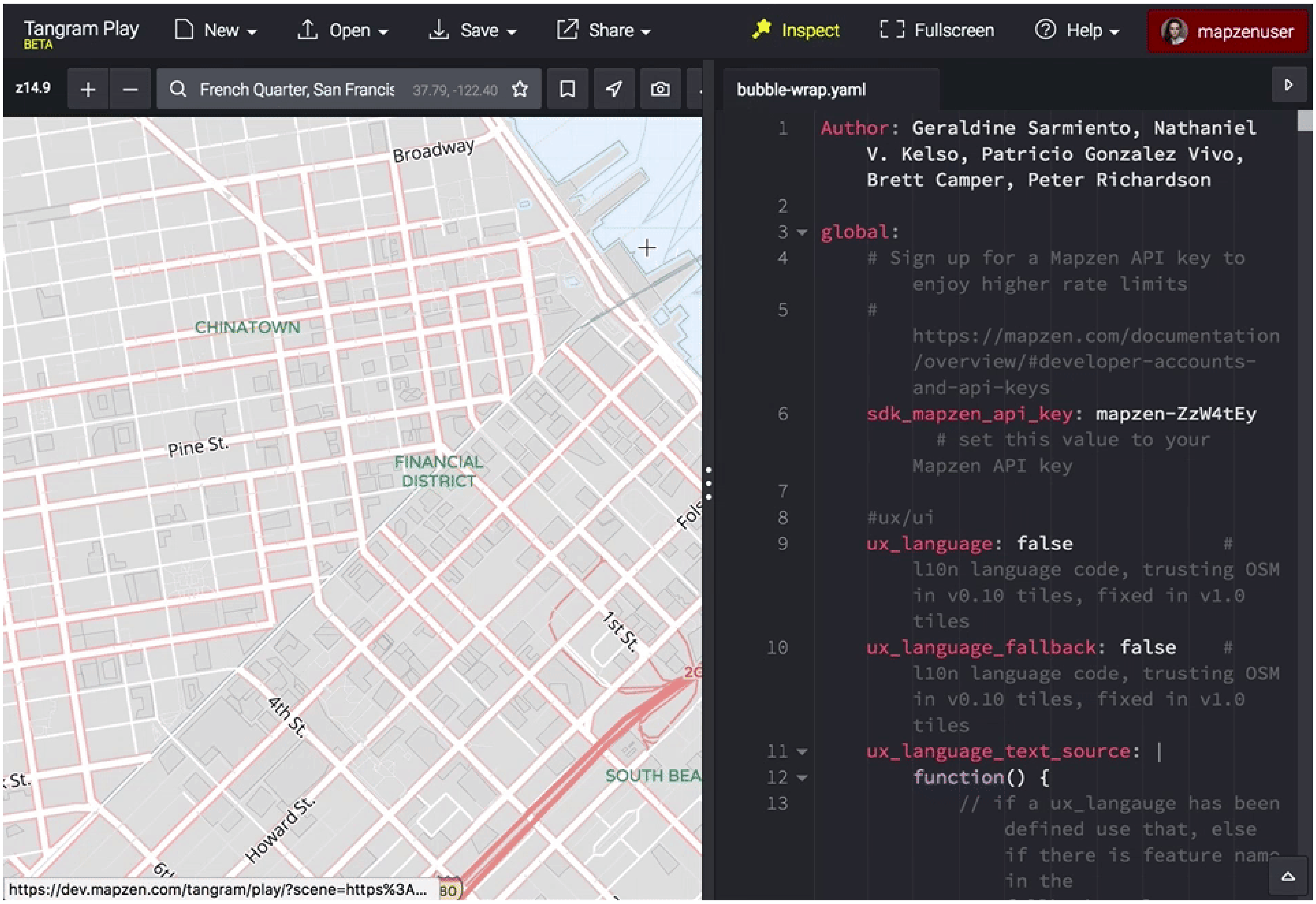

Tangram Play
BETA
New
Open
Save
Share
Inspect
Fullscreen
Help
mapzenuser
z13.5
Lower Manhattan, Manhatta
40.72, -73.99
basic.yaml
HEX
R
G
B
A

You have unsaved changes.

```
                4px], [17,
                10m]]
        link:
            filter: { is_link: true
                } # on- and off-
                ramps, etc
            draw:
                lines:
                    width:
                        [[8,0px],
                        [14, 3px],
                        [16, 5px],
                        [18, 10m]]
    tunn
    building
        data
        draw
            color: [0.094, 0.773,
                0.847, 1.00]
            extrude: true
```

HEX	R	G	B	A
18C5D8	24	197	216	1.00

- Inspect each pixel on the map to see the original tile source data that generated it, helping you improve you understand and improve your styles.

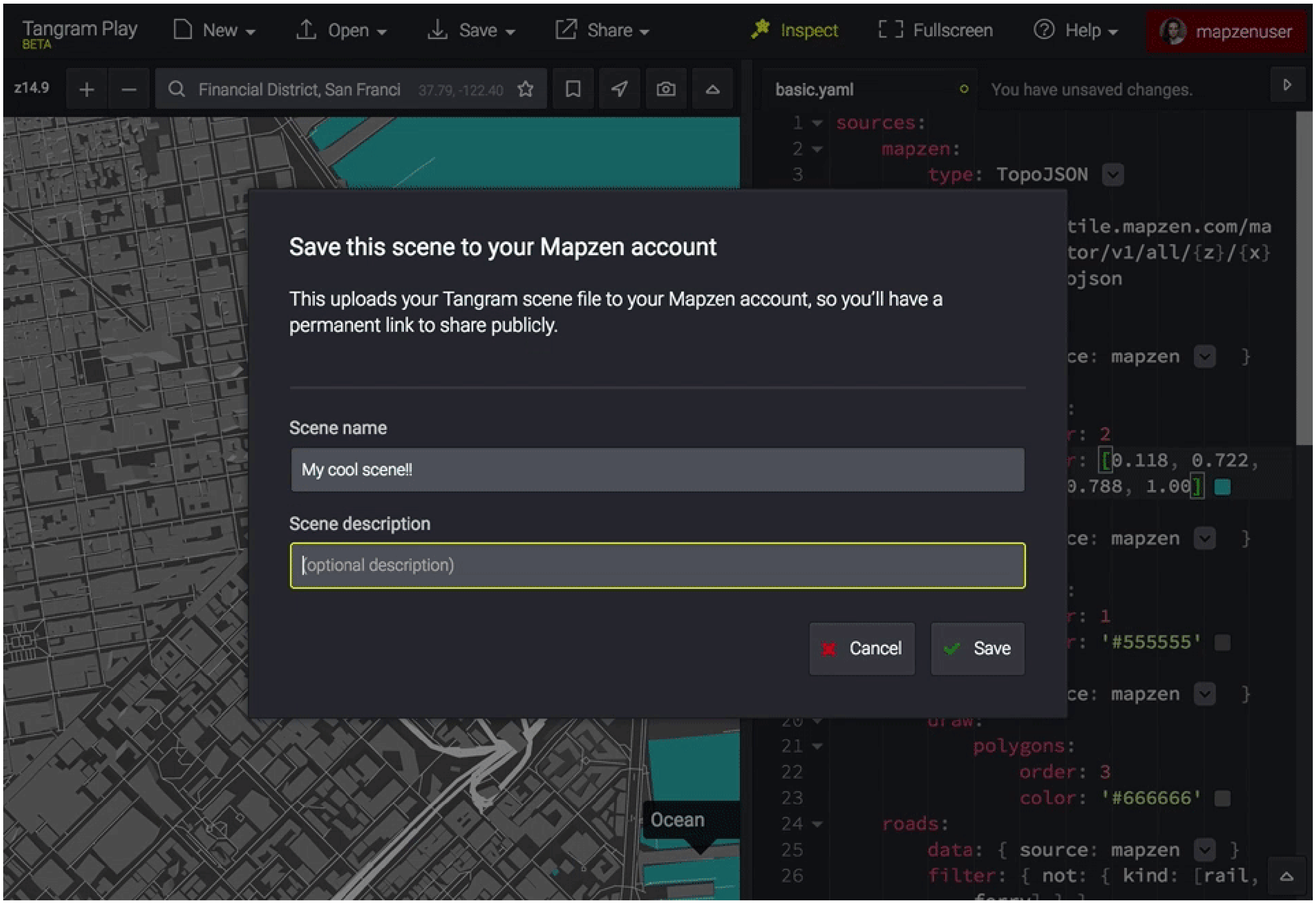

Tangram Play
BETA
New
Open
Save
Share
Inspect
Fullscreen
Help
mapzenuser
z14.9
French Quarter, San Francis
37.79, -122.40
bubble-wrap.yaml
Broadway
CHINATOWN
Pine St.
FINANCIAL DISTRICT
1st St.
4th St.
Howard St.
SOUTH BEA
https://dev.mapzen.com/tangram/play/?scene=https%3A...

```yaml
Author: Geraldine Sarmiento, Nathaniel V. Kelso, Patricio Gonzalez Vivo, Brett Camper, Peter Richardson

global:
    # Sign up for a Mapzen API key to enjoy higher rate limits
    # https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys
    sdk_mapzen_api_key: mapzen-ZzW4tEy  # set this value to your Mapzen API key

    #ux/ui
    ux_language: false          # l10n language code, trusting OSM in v0.10 tiles, fixed in v1.0 tiles
    ux_language_fallback: false    # l10n language code, trusting OSM in v0.10 tiles, fixed in v1.0 tiles
    ux_language_text_source: |
        function() {
            // if a ux_langauge has been defined use that, else if there is feature name in the
```

You can learn even more about Tangram Play's interface in our **Getting Started (https://mapzen.com/tangram/play/docs/)** document.

New stuff!

But wait, there's more! With this launch we are rolling out a completely new feature: the ability to save your scenes to your Mapzen account! Once you've created a scene file, you can either download it to your own computer, or save it to the cloud. You get a link to share your map with your friends, along with a snippet of Javascript so you can embed the scene file into your own website.


Tangram Play
BETA
New
Open
Save
Share
Inspect
Fullscreen
Help
mapzenuser
z14.9
Financial District, San Franci 37.79, -122.40
basic.yaml
You have unsaved changes.
Ocean

Save this scene to your Mapzen account

This uploads your Tangram scene file to your Mapzen account, so you'll have a permanent link to share publicly.

Scene name

My cool scene!!

Scene description

(optional description)

Cancel

Save

We want to hear from you!

We understand that there are many different kinds of maps, and many different kinds of map makers. While no single tool can meet everyone's needs, our goal is to create a friendly and educational developer environment that is useful for everyone, whether beginner or expert. To get there, we'll be listening to you. So please reach out to us at a **(mailto:tangram@mapzen.com?subject=[Tangram%20Play])tangram@mapzen.com (mailto:tangram@mapzen.com)** if you have any feedback or questions for us. You can also join us at **Gitter (https://gitter.im/tangrams/tangram-chat)**, and we also welcome your contributions or issues on Tangram Play's **GitHub repository (https://github.com/tangrams/tangram-play/)**, where all the code is open for you to explore.

Happy playing!

· 07 February 2017 ·

Lou Huang

Lou is a protagonist of an open world.

© 2017 Mapzen

Bundling up descendants into GeoJSON

whosonfirst (/tag/whosonfirst) data (/tag/data)

Every record in **Who's On First (https://whosonfirst.mapzen.com/spelunker/)**, our gazetteer of places around the world, has two powerful properties – geometry and hierarchy. Since the raw data of any WOF record is a blob of GeoJSON, you can take the URL and draw it in any modern mapping application with minimal effort. We thought it would be nice to let you grab GeoJSON for a bunch of records at once, so we added a tool called the Bundler for you using the soon-to-be-released Who's On First API.

Descending into Mapness

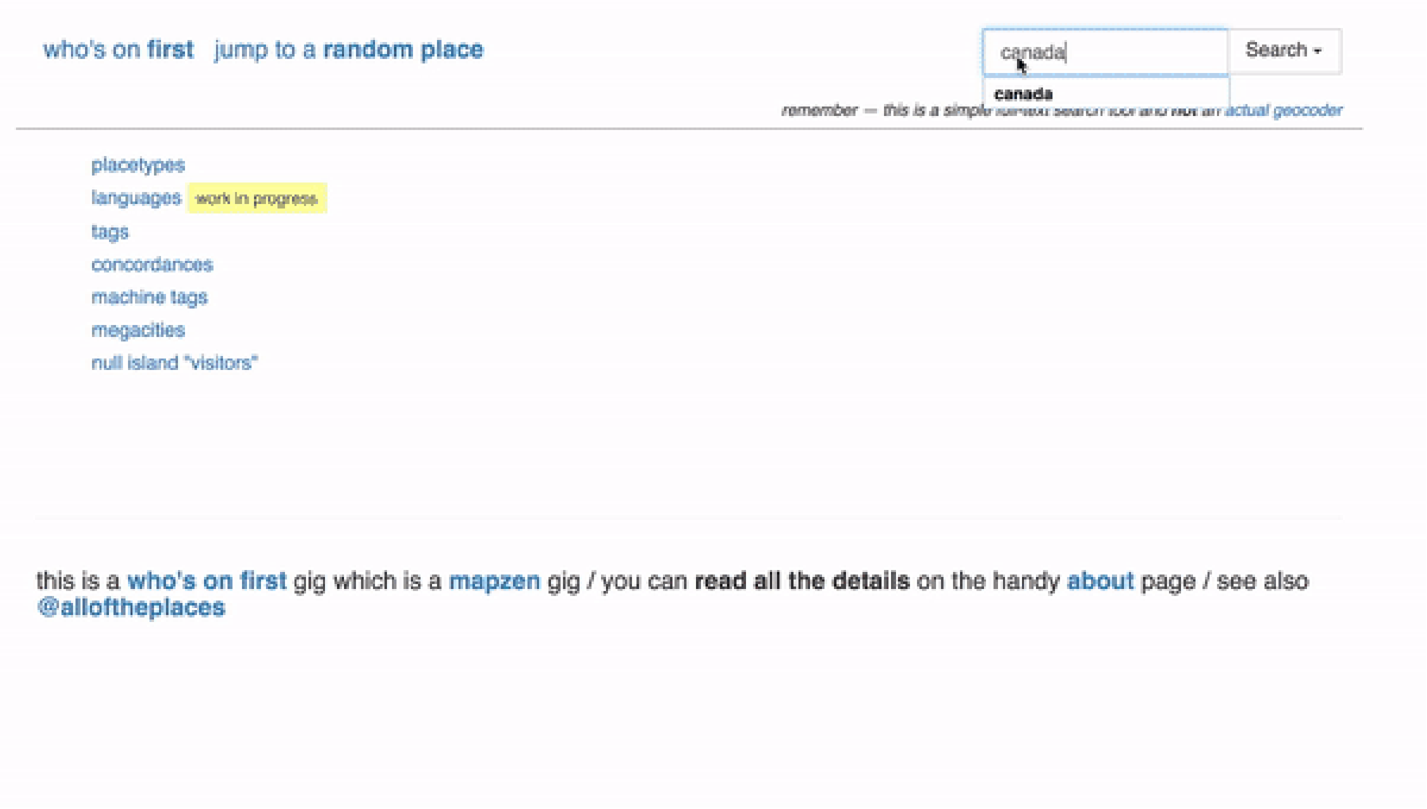
Say you want to highlight **WOF ID 85682085 (https://whosonfirst.mapzen.com/spelunker/id/85682085/#4/54.87/-95.50)** on a map. This is the permanent ID of Manitoba, home of **The Weakerthans (https://www.youtube.com/watch?v=AJBEVgnND6k)** and a province with a **cool step function for its border (http://www.theglobeandmail.com/opinion/why-the-zigzag-between-sask-and-man/article4202831/)** with **Saskatchewan (https://whosonfirst.mapzen.com/spelunker/id/85682113/#4/54.87/-105.68)**.

You can add the URL for the raw data – **https://whosonfirst.mapzen.com/data/856/820/85/85682085.geojson (https://whosonfirst.mapzen.com/data/856/820/85/85682085.geojson)** – as a source **in Tangram Play and have a map in seconds (https://mapzen.com/tangram/play/?scene=https://s3.amazonaws.com/whosonfirst.mapzen.com/misc/_blogs/descender_one.yaml&lines=11-13#5.000/55.292/-97.734)**:

But what if you wanted to download *all* the regions of a country? You could grab the URL for **Saskatchewan (https://whosonfirst.mapzen.com/spelunker/id/85682113/#4/54.87/-105.68)** and then **Alberta (https://whosonfirst.mapzen.com/spelunker/id/85682091/#18/48.99668/-120.00138)** and then **British Columbia (https://whosonfirst.mapzen.com/spelunker/id/85682117/#18/48.22455/-139.06111)** and then the **Yukon (https://whosonfirst.mapzen.com/spelunker/id/85682095/#2/71.3/-96.8)**… or you could leverage **the hierarchy (https://whosonfirst.mapzen.com/placetypes/)** that’s included in every Who’s On First record.

In the record for Manitoba, there’s a list of relations, both upward (Canada, North America) and downward (localities, postal codes, venues…) And if you look at **Canada’s record (https://whosonfirst.mapzen.com/spelunker/id/85633041/descendants/?exclude=nullisland&placetype=region#3/60.64/-114.45)**, you see all the descendants that are regions, including Manitoba.

Since these relationships are all available via the Who's On First API, we decided to implement a tool we're calling the Bundler, based off an experiment called the **Descender (https://burritojustice.github.io/wof-descender/)**. It lets you collect all the geometries of specified levels of the descendants of a parent and bundles them into one handy blob of GeoJSON for you to download and use as you see fit.


who's on first
jump to a random place
canada
Search
canada
remember — this is a simple
actual geocoder
placetypes
languages
work in progress
tags
concordances
machine tags
megacities
null island "visitors"

this is a who's on first gig which is a mapzen gig / you can read all the details on the handy about page / see also @alloftheplaces

Life Before Bundles

(Stephen here.)

Prior to creation of the Bundler and the **Descender (https://burritojustice.github.io/wof-descender/)** (discussed below), our workflow to gather feature collections was much more complicated, requiring fairly laborious and tedious steps:

- Clone the **whosonfirst-data repository (https://github.com/whosonfirst-data/whosonfirst-data)** (tens of GB of data)
- Clone the **py-mapzen-whosonfirst-utils repository (https://github.com/whosonfirst/py-mapzen-whosonfirst-utils)**
- Gather a list of `wof:id` values in a single CSV file
- Run the following command, invoking the **`csv-to-feature-collection` script (https://github.com/whosonfirst/py-mapzen-whosonfirst-**

utils/blob/master/scripts/wof-csv-to-feature-collection) in the py-mapzen-whosonfirst-utils repository:

```
python wof-csv-to-feature-collection -p /usr/local/mapzen/whosonfirst-data/data
-c ~/Desktop/list_of_wofids.csv -o ~/Desktop/featurecollection.geojson full -i
```

Even after cloning the repo, the method of locally gathering a feature collection took a long time - a typical bundle of a few dozen descendants would take 2 to 3 minutes to get ready, and then another 3 minutes to generate, so we needed 5 to 6 minutes per bundle. This process was more time consuming than it needed to be and made us, well, sad; hence the birth of the Descender.

The Descender

(John here.)

After talking to Aaron and Stephen about getting descendants out of WOF, I decided to build a proof of concept using the WOF API and wired together the **Descender (https://burritojustice.github.io/wof-descender/)** in Javascript. In order to gather and group a set of descendants from a parent, it…

- searches for a Who's On First ID
- lets you select an arbitrary level of descent, whether they are directly related (regions of a country, counties of a region) or skipping levels of hierarchy (counties of a country, regions of a continent)
- calls up the list of the parents descendants for that level, and records their WOF IDs
- constructs URLs for the GeoJSON files for each descendant
- downloads a descendant, draws it on a map, and adds that GeoJSON into an array
- converts that array into a GeoJSON feature collection and stores it as a blob
- uses **FileSaver.js (https://github.com/eligrey/FileSaver.js/)** to save the blob to the desktop, complete with a descriptive name.

While the search is clunky and it only gathers the first 500 descendants, it is fast, and actual GeoJSON appears on your desktop! We started using it right away in QGIS and Tangram.

That 5 to 6 minute process that we mentioned? It's no exaggeration to say it can take just 5 to 6 seconds in the Descender. This made Stephen (and others) happy.

Bringing the Descender into the Spelunker as the Bundler

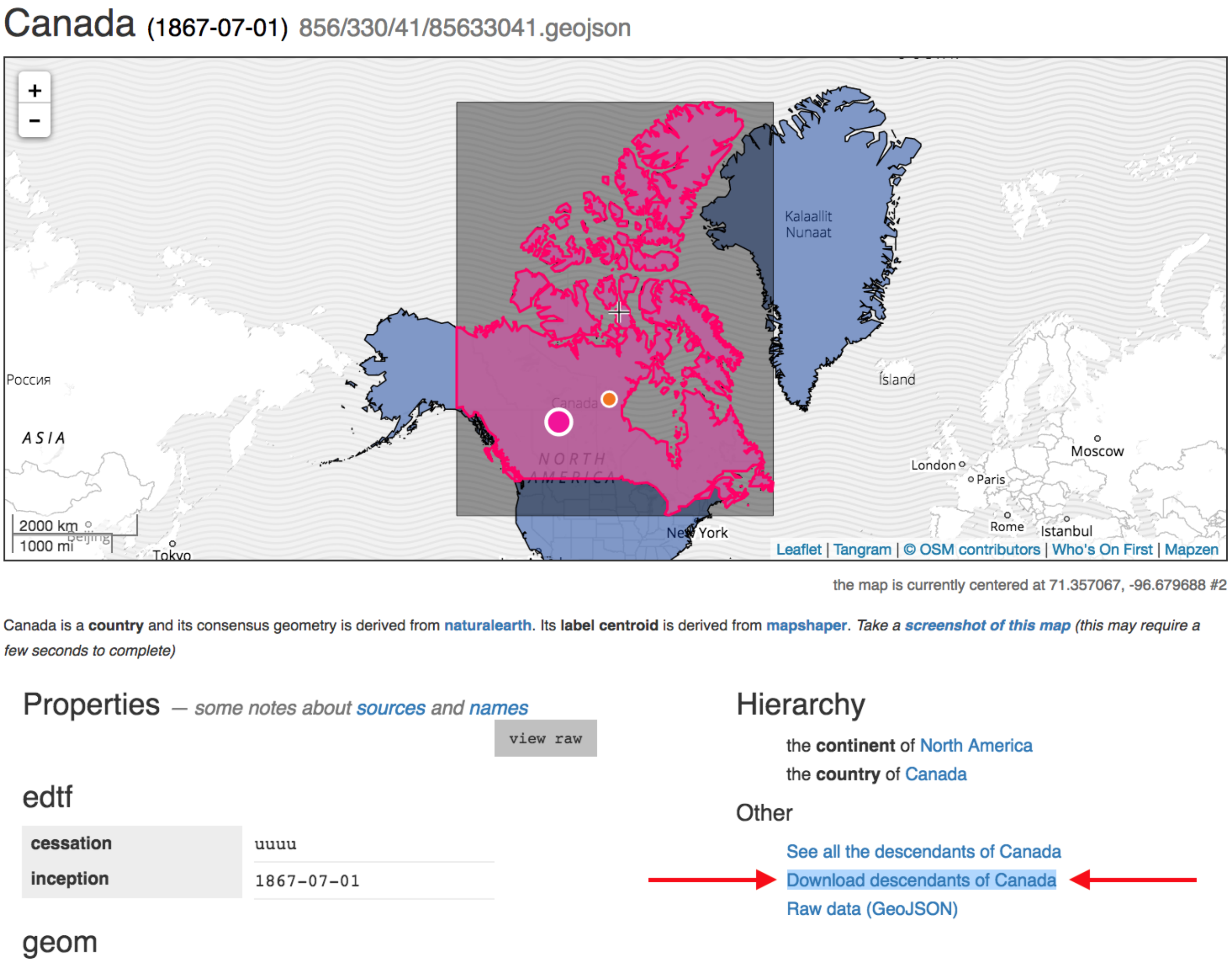
(Dan here.)

When I saw John's Descender tool, I knew this should be integrated into the **Who's On First Spelunker (https://burritojustice.github.io/wof-descender/)**, which is how most of us explore and discover records for places. Along with Stephen, we began by copying the code wholesale, without regard for the existing site templates. Then I started to dig into the JavaScript (all of the data bundling happens in-browser on the front-end).

The process of refactoring the Descender involved reorganizing & tidying the code, breaking it into smaller self-contained functions, isolating variable namespaces, and separating the **visible UI parts (https://github.com/whosonfirst/whosonfirst-www-spelunker/blob/master/www/static/javascript/mapzen.whosonfirst.spelunker.init.download.js)**, which are inherently specific to a single web page, from **the "business logic" (https://github.com/whosonfirst/whosonfirst-www-spelunker/blob/master/www/static/javascript/mapzen.whosonfirst.bundler.js)** that controls the **querying (https://github.com/whosonfirst/whosonfirst-www-spelunker/blob/14d8fd2d0f926c6e2679c8c0fac02d3c082202ae/www/static/javascript/mapzen.whosonfirst.bundler.js#L166)**, **downloading (https://github.com/whosonfirst/whosonfirst-www-spelunker/blob/14d8fd2d0f926c6e2679c8c0fac02d3c082202ae/www/static/javascript/mapzen.whosonfirst.bundler.js#L227)** and **bundling (https://github.com/whosonfirst/whosonfirst-www-spelunker/blob/14d8fd2d0f926c6e2679c8c0fac02d3c082202ae/www/static/javascript/mapzen.whosonfirst.bundler.js#L97)** mechanisms. This separation of concerns means that we have a general purpose WOF downloading tool that can be bolted onto a variety of other kinds of pages. It means you could potentially add a WOF bundler into your own specific mapping page.

On a very high level the code asks a REST API endpoint for a list of WOF IDs of a `placetype` that share a common ancestor record, then it just downloads them all into memory. When all the downloads finish the code assembles the accumulated blob of data into the flexible GeoJSON file format, and saves it to disk thanks to Eli Grey's **FileSaver.js library (https://github.com/eligrey/FileSaver.js/)**.

Once we got the Descender working as-is, we started wiring in links from various pages in the Spelunker (for example, in the "Other Links" section of a record).



Canada (1867-07-01) 856/330/41/85633041.geojson

+
-
Kalaallit
Nunaat
Россия
ASIA
Canada
NORTH
AMERICA
Ísland
Moscow
London
Paris
Rome
Istanbul
New York
Beijing
Tokyo
2000 km
1000 mi
Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen

the map is currently centered at 71.357067, -96.679688 #2

Canada is a **country** and its consensus geometry is derived from **naturalearth**. Its **label centroid** is derived from **mapshaper**. *Take a* ***screenshot of this map*** *(this may require a few seconds to complete)*

Properties — *some notes about sources and names*

view raw

edtf

cessation	uuuu
inception	1867-07-01

geom

Hierarchy

the **continent** of North America
the **country** of Canada

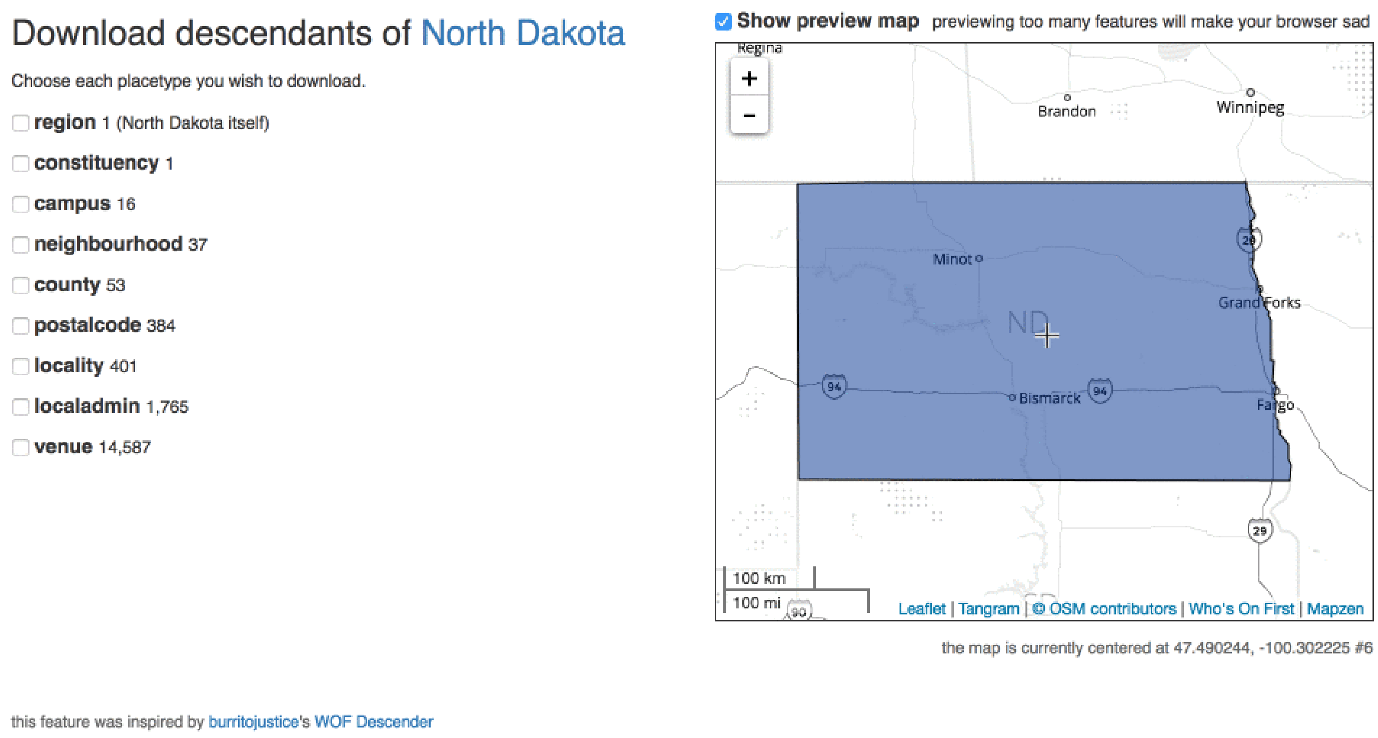
Other

See all the descendants of Canada
Download descendants of Canada
Raw data (GeoJSON)

What can you do with the Bundler?

From the perspective of a user of data, the benefits of the Bundler are substantial; if a user would like to contribute to updating neighbourhood features in Who's On First (outlined in a **previous blogpost (https://mapzen.com/blog/sf-neighbourhood-updates/)**) or gather collections of new county features (outlined in another **previous blogpost (https://mapzen.com/blog/mesoshapes/)**), a few simple clicks from the **Spelunker (https://whosonfirst.mapzen.com/spelunker/)** are all it takes to get started!

We also let you gather descendants for different placetypes at the same time – for example, you can get the counties and postal codes of North Dakota all bundled up in one handy blob of GeoJSON.



Download descendants of North Dakota

Choose each placetype you wish to download.

- ☐ **region** 1 (North Dakota itself)
- ☐ **constituency** 1
- ☐ **campus** 16
- ☐ **neighbourhood** 37
- ☐ **county** 53
- ☐ **postalcode** 384
- ☐ **locality** 401
- ☐ **localadmin** 1,765
- ☐ **venue** 14,587

☑ **Show preview map** previewing too many features will make your browser sad

Regina
+
−
Brandon
Winnipeg
Minot
ND
Grand Forks
94
Bismarck
94
Fargo
29
29
100 km
100 mi
90
Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen

the map is currently centered at 47.490244, -100.302225 #6

this feature was inspired by burritojustice's WOF Descender

A few examples of what you can collect through the Bundler:

The regions of South America

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

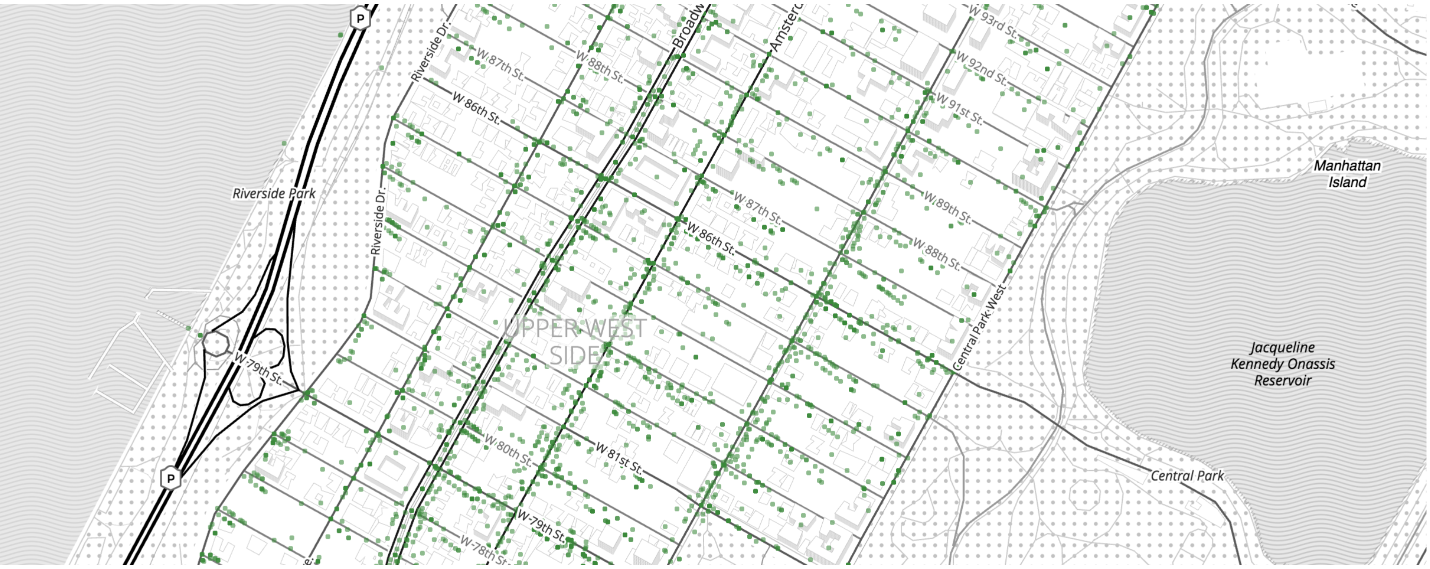
Open in Tangram Play! (https://mapzen.com/tangram/play/?noscroll&maxbounds=-62.35,-135.53,32.10,8.44&scene=https://s3.amazonaws.com/whosonfirst.mapzen.com/misc/_blogs/descender_two.yaml#3.18/-22.62/-63.29)

The regions of Ireland

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Open in Tangram Play! (https://mapzen.com/tangram/play/?noscroll&maxbounds=51.94,-14.60,54.99,-1.38&scene=https://s3.amazonaws.com/whosonfirst.mapzen.com/misc/_blogs/descender_three.yaml#8.9835/53.3480/-7.7008)

9,000 venues in the Upper West Side of New York City


W 93rd St.
W 92nd St.
W 91st St.
Broadway
Amsterdam
Riverside Dr.
W 87th St.
W 88th St.
W 86th St.
Riverside Park
Riverside Dr.
W 87th St.
W 89th St.
W 86th St.
W 88th St.
Manhattan Island
Central Park West
UPPER WEST SIDE
W 79th St.
Jacqueline Kennedy Onassis Reservoir
W 80th St.
W 81st St.
Central Park
W 79th St.
W 78th St.

(click map to load 9000 points, and zoom in@)

Open in Tangram Play! (https://mapzen.com/tangram/play/?noscroll&minz=13&maxbounds=40.769,-74.031,40.803,-73.918&scene=https://s3.amazonaws.com/whosonfirst.mapzen.com/misc/_blogs/descender_four.yaml#15.825/40.7866/-73.9737)

You can also use it to update neighbourhood records for your locality, **as we discuss in our guide here (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/issue_workflows/sf_neighbourhood_updates_pt_1.md)**.

Because the GeoJSON contains the entire record, you can pull out and display on the map any data values you see in WOF, including alternative language place names and population as we did with the **regions of India (https://mapzen.com/blog/languages-of-india/)**.

Things you should know, and things we should know

Take the Bundler for a spin! A few notes and caveats:

- You may notice slightly offset geometries across placetypes; if this occurs, please **file an issue (https://github.com/whosonfirst-data/whosonfirst-data/issues/)** so we can actively track these updates.
- Beware that downloading very large bundles can be a memory-intensive process and can make your web browser… sad. We've successfully tested it on some fairly big descents, including the **counties of the United States (https://mapzen.com/blog/administrivia-counties/)**, which is 3100+ records and over 180MB of GeoJSON. (We also learned the hard way that we needed to turn off the preview map for bundles over 100MB or your browser would get… sad. We try do this for you automatically for big bundles, but you can toggle it yourself if things start to get slow.)
- There's a limit of 10,000 descendants in a bundle for the time being – we're working on increasing that, and we're also looking into building a command line tool, so stay tuned! If one of your bundles fails, **let us know (mailto:hello@mapzen.com?subject=bundler%20is%20sad)**.
- In October 2016 we retired **Borders (https://mapzen.com/data/borders)**, a tool for downloading administrative boundary data from OpenStreetMap. You can readily access equivalent hierarchies and boundaries in Who's On First by searching for a country and

following its download descendants link. Here’s a **list of concordances between Borders and WOF (https://whosonfirst.mapzen.com/spelunker/concordances/mapzenborders/#)**.

- Another thing that might be useful are **placetype bundles (https://whosonfirst.mapzen.com/bundles/)** if you want to do a bulk download of WOF data by placetype.
- The Who’s On First API is a thing but it’s not quite ready for general-public release (but will be soon).

But most importantly, **let us know (mailto:hello@mapzen.com?subject=bundler%20is%20happy)** how you use the Bundler!

· 10 February 2017 ·

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.

Dan Phiffer

Dan is a data carpenter, which tbh is kind of a made up title. But he likes it all the same.

© 2017 Mapzen

Customize Your Search

mapzen-js (/tag/mapzen-js) demo (/tag/demo) tutorial (/tag/tutorial) search (/tag/search) geocoding (/tag/geocoding) make-your-own (/tag/make-your-own)

Hello!

Welcome to another **Make Your Own (https://mapzen.com/tag/make-your-own/)** [Map].

Today I'm going to take a break from Tangram cartography to talk about another feature of **mapzen.js (https://github.com/mapzen/mapzen.js)**: **the geocoder!**

Or, yeah, you can call it the search box. 🙂

The **mapzen.js geocoder (https://mapzen.com/documentation/mapzen-js/search/)** is built on **Mapzen Search (https://mapzen.com/documentation/search/)**, a geocoding and place-finding service.

We built the geocoder with sane defaults, so that it will work for your map straight out-of-the-box. However, there are times when you might want to change the look and interaction of the search box, or even tweak the search query itself.

To show you some of the customizations that we can make to the geocoder, I'm going to walk through two different examples.

Let's start with a quick campus map.

Campus Map

Open up your favorite text editor and create an `index.html` file we can view in our browser:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Campus Map | The mapzen.js geocoder</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      html,body{margin: 0; padding: 0}
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
  </style>
  </head>
<body>
  <div id="map"></div>

  <script>
    // Mapzen API key (replace key with your own)
    // To generate your own key, go to https://mapzen.com/developers/
    L.Mapzen.apiKey = 'mapzen-JA21Wes';

    var umn = [44.9745, -93.2346];
    var southWest = L.latLng(44.96289, -93.24696),
        northEast = L.latLng(44.99170, -93.16660),
        umnBounds = L.latLngBounds(southWest, northEast);

    var map = L.Mapzen.map('map', {
      center: umn,
      zoom: 16,
      tangramOptions: {
        scene: L.Mapzen.BasemapStyles.CinnabarMoreLabels
      }
    });

    // Instantiate the geocoder (a.k.a the search box)
    var geocoder = L.Mapzen.geocoder();

    // Add geocoder to map
    geocoder.addTo(map);

  </script>
</body>
</html>
```

Need help getting started? *Check out our* ***guide to setting up a bare bones development environment (https://mapzen.com/documentation/guides)****.*

API Key

The first thing you may have noticed is the need for an API key. The API key is a code that identifies your Mapzen developer account without you needing to provide a password. Handy!

The API key isn't unique to just the geocoder. All of Mapzen's hosted services are shared resources, so we require an API key to prevent individual users from degrading system performance for everyone.

See the ***full list of Mapzen services and rate limits (https://mapzen.com/documentation/overview/#search-and-mobility-products)****.*

To make API key juggling a little easier, **mapzen.js (https://github.com/mapzen/mapzen.js)** allows you to set a single global key that will be used for all Mapzen services:

```
L.Mapzen.apiKey = 'mapzen-JA21Wes';
```

And don't worry–sharing the same API key between different services will not affect individual rate limits. They don't share limits.

The API key we're using here (`'mapzen-JA21Wes'`) is used by many of our demos. It's fine to use it while working through these examples, but keep in mind that rate limits will be shared by everyone using that same key. So, if you find that tiles aren't loading properly, then it's probably time to snag your own. (Also, we'll be rotating this demo key every few months, so don't get too attached to it…)

Sign up for a free API key at https://mapzen.com/developers (https://mapzen.com/developers).

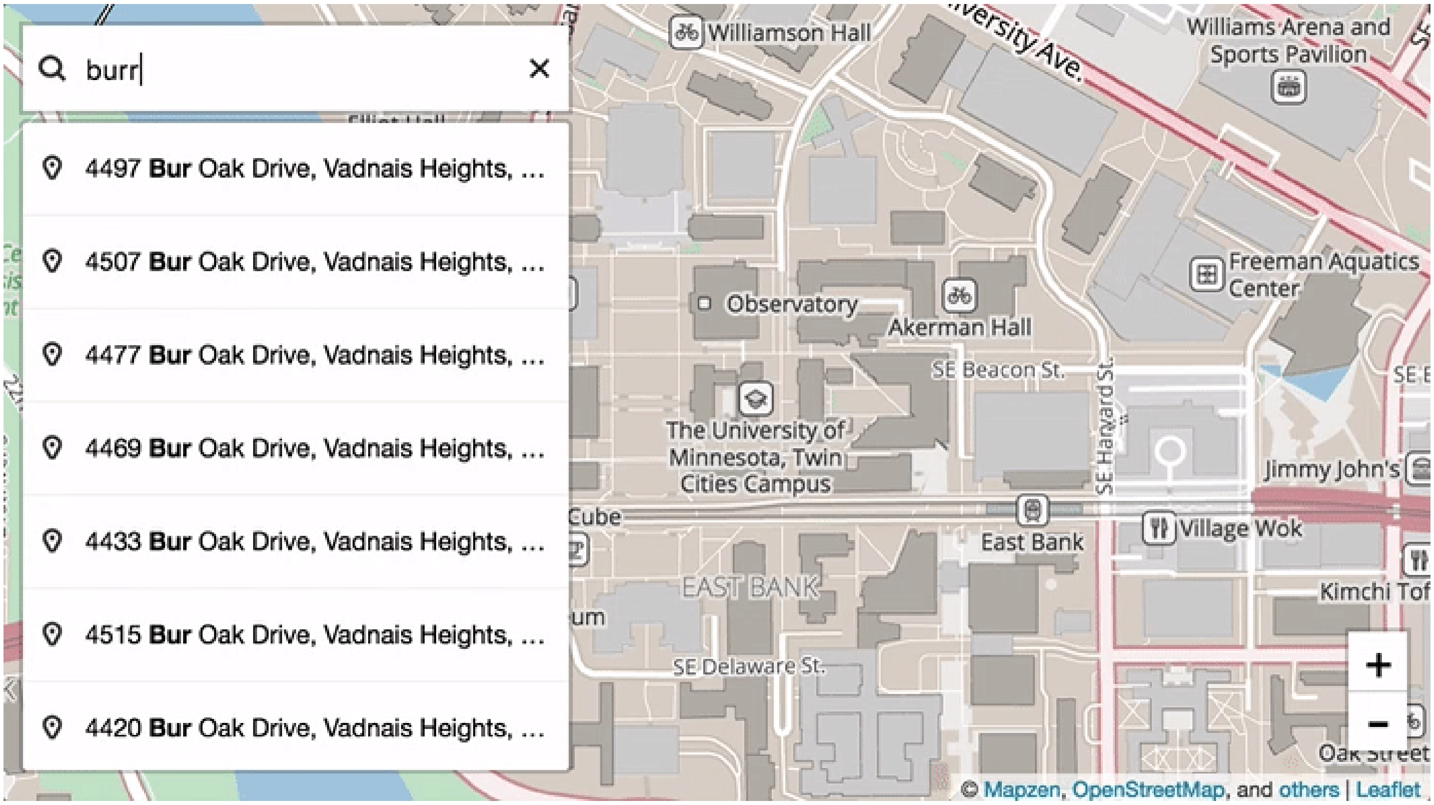
And before you ask… Yes! Mapzen offers a generous free tier, which should be plenty for small projects. If it's not, and you find yourself bumping up against rate limits, just **let us know (mailto:hello@mapzen.com)**.

Ok. Back to our campus map.

So far we have a simple map zoomed in on the University of Minnesota Twin Cities (*go Gophers!*), along with the default search box.

Go ahead and give it a spin.

If we start typing in "burrito", you'll see that we immediately get a list of possible search results:



burr

- 4497 **Bur** Oak Drive, Vadnais Heights, ...
- 4507 **Bur** Oak Drive, Vadnais Heights, ...
- 4477 **Bur** Oak Drive, Vadnais Heights, ...
- 4469 **Bur** Oak Drive, Vadnais Heights, ...
- 4433 **Bur** Oak Drive, Vadnais Heights, ...
- 4515 **Bur** Oak Drive, Vadnais Heights, ...
- 4420 **Bur** Oak Drive, Vadnais Heights, ...

Williamson Hall
University Ave.
Williams Arena and Sports Pavilion
Freeman Aquatics Center
Observatory
Akerman Hall
SE Beacon St.
SE Harvard St.
The University of Minnesota, Twin Cities Campus
Jimmy John's
Cube
East Bank
Village Wok
EAST BANK
Kimchi Tof
SE Delaware St.
Oak Street
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

Gonna find me a burrito

Some of these are relevant… Burrito Loco is certainly nearby. And despite El Burrito Mercado being all the way over in St. Paul, I am a fan so I'll let it slide.

Wait, Holmen, WI? Where?

AMES, IOWA?!

Ok, clearly this is not the most relevant list of locations for our late-night burrito run.

Not to worry, we can fix this. There are few **burrito problems (http://idlewords.com/2007/04/the_alameda_weehawken_burrito_tunnel.htm)** that **Mapzen can't solve (https://mapzen.com/tangram/view/?**

scene=https://mapzen.com/api/scenes/22/167/resources/empty.yaml#4.817/39.433/-95.544).

Let’s try filtering our results.

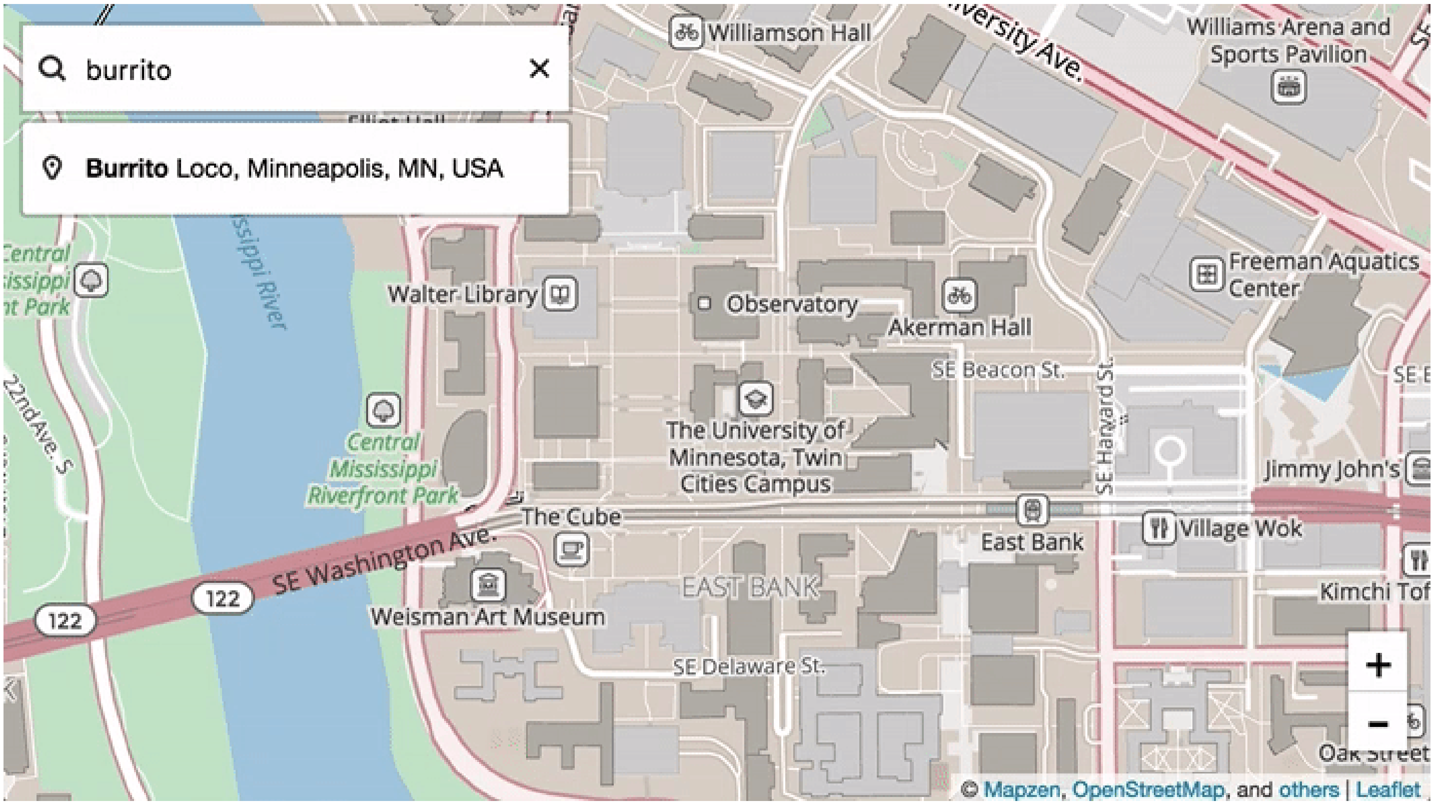
Filtering results

By default, the geocoder prioritizes results found near the center of our current map view. If you want, you can turn this off by setting `focus: false` . Of course, we don’t want to do that in this example. In fact, we’d like to go one step further and **limit our results to a defined bounding box**.

We already have a bounding box defined in our initial `index.html` file called `umnBounds` . Let’s use that bounding box to set the `bounds` option on our geocoder:

```
    // Define geocoder options
    var geocoderOptions = {
      bounds: umnBounds
    };

    // Instantiate the geocoder (a.k.a the search box)
    var geocoder = L.Mapzen.geocoder(geocoderOptions);
```

Now, when we run our same burrito search…



burrito

Burrito Loco, Minneapolis, MN, USA

Williamson Hall
Williams Arena and Sports Pavilion
Walter Library
Observatory
Akerman Hall
Freeman Aquatics Center
SE Beacon St.
SE Harvard St.
Central Mississippi Riverfront Park
The University of Minnesota, Twin Cities Campus
Jimmy John's
The Cube
East Bank
Village Wok
SE Washington Ave.
122
Weisman Art Museum
EAST BANK
Kimchi Tof
SE Delaware St.
22nd Ave. S
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

*Gonna find me a **nearby** burrito*

Ok. I'll admit. It's sad that only one burrito place is showing up. But also… it's Minnesota. Perhaps I should have expected that.

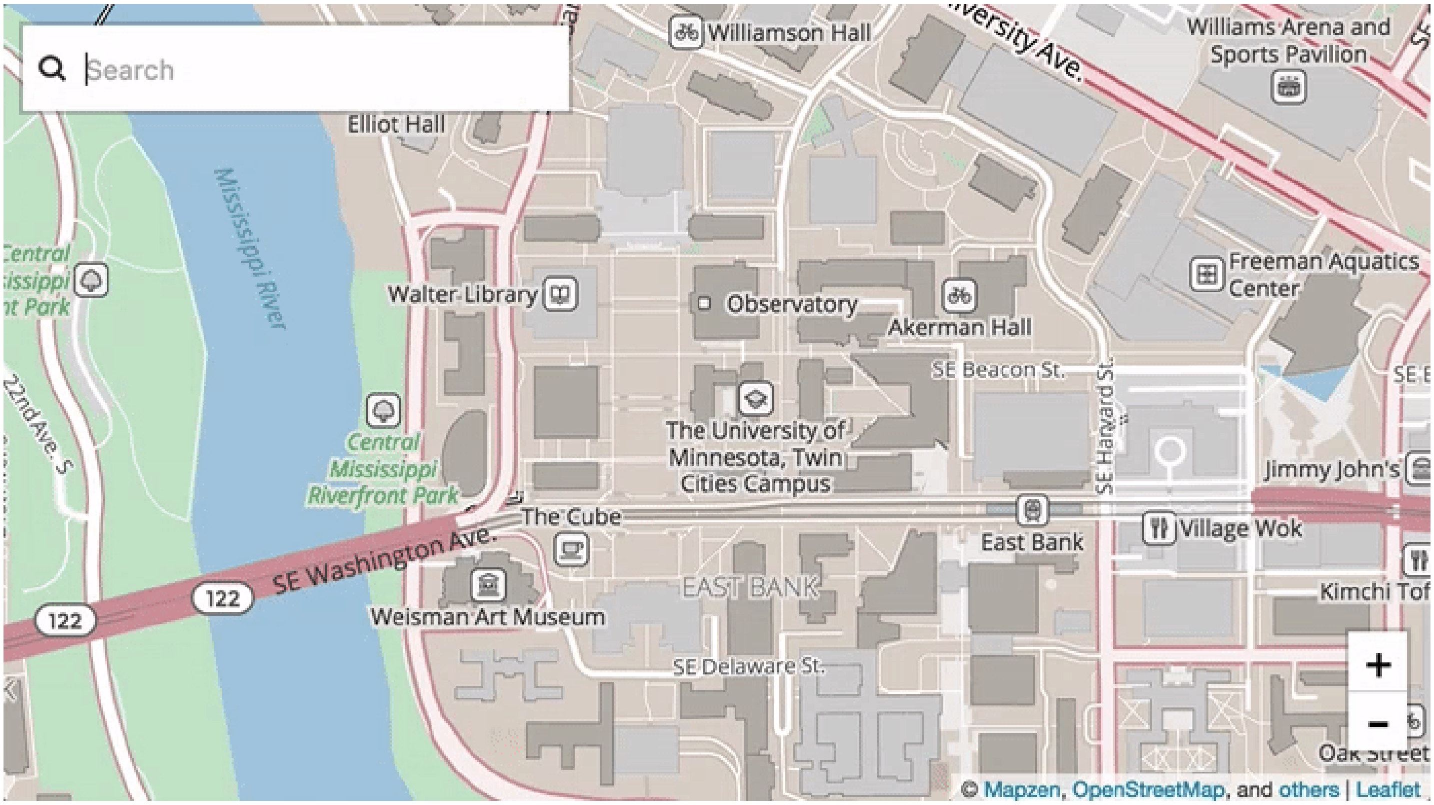
In happier news, our bounded search works!

Excellent!

Next, I want to look up one of my favorite buildings on campus (and my home away from home back in the twenty-oughts): Pillsbury Hall.

Pillsbury Hall, the second oldest building on the Twin Cities campus. Home to rock samples, stressed-out grad students, and the occasional mouse.

Let’s start typing…


Search
Williamson Hall
Williams Arena and Sports Pavilion
Elliot Hall
Mississippi River
Walter Library
Observatory
Akerman Hall
Freeman Aquatics Center
SE Beacon St.
SE Harvard St.
22nd Ave. S
Central Mississippi Riverfront Park
The University of Minnesota, Twin Cities Campus
Jimmy John's
The Cube
East Bank
Village Wok
SE Washington Ave.
122
122
EAST BANK
Kimchi Tof
Weisman Art Museum
SE Delaware St.
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

Huh. Well, it found the building. But there are also a lot of addresses that came up. That's not so helpful. Let's filter those out using the **layers option (https://mapzen.com/documentation/mapzen-js/search/#control-the-search-query-behavior)**.

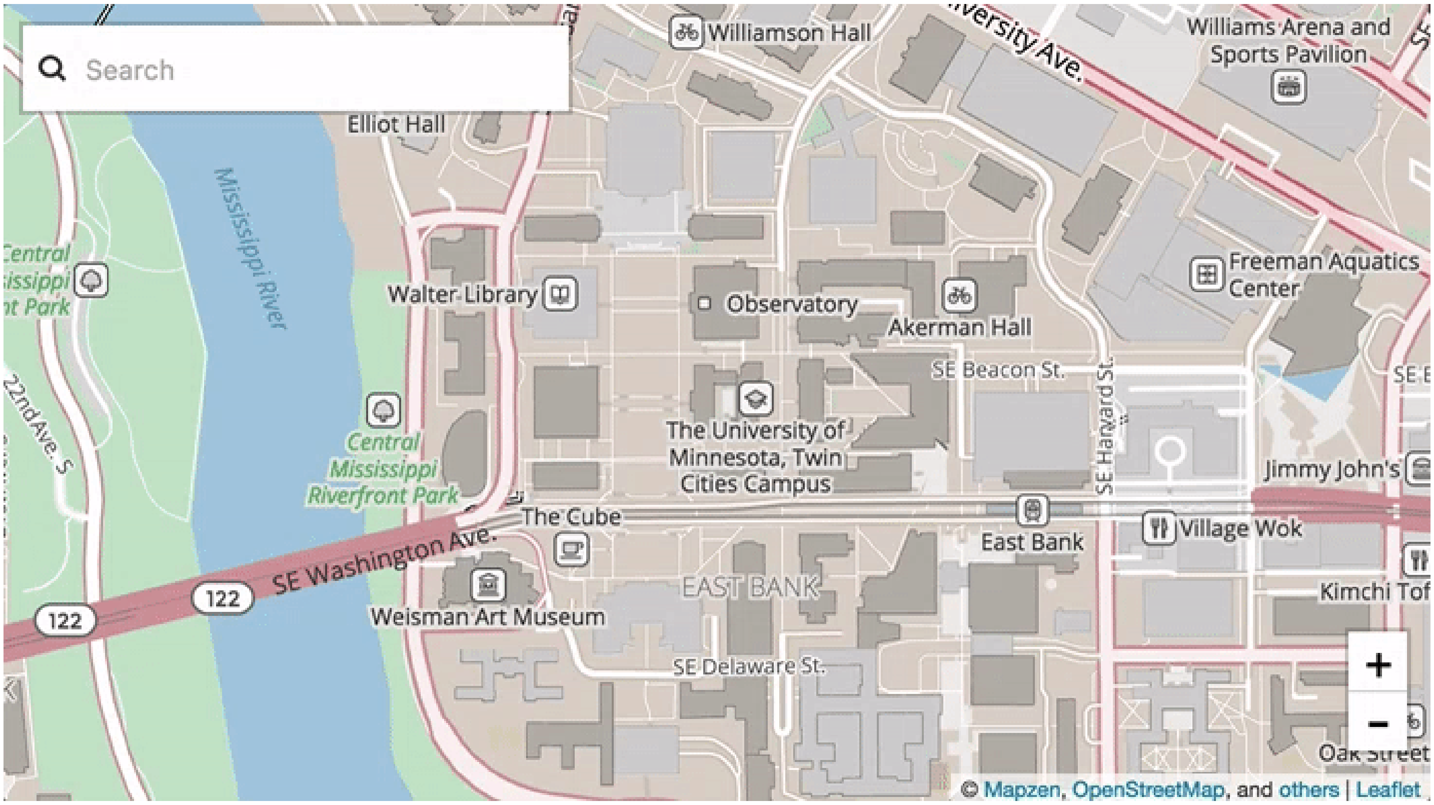
The `layers` option **limits the type of places the geocoder will search**. This will allow you to focus on everything from fine-resolution geocoding (like venues and addresses) to coarse geocoding (neighborhoods, counties, and even countries).

*See the **full list of place types available to search (https://mapzen.com/documentation/search/search/#filter-by-data-type)**.*

Let's limit our search results to just "venues":

```
var geocoderOptions = {
  bounds: umnBounds,
  layers: ['venue']
};
```

Now, when we search for "Pillsbury Hall", our search results no longer include addresses.


Search
Williamson Hall
University Ave.
Williams Arena and Sports Pavilion
Elliot Hall
Mississippi River
Central Mississippi Riverfront Park
Walter Library
Observatory
Akerman Hall
Freeman Aquatics Center
SE Beacon St.
SE Harvard St.
The University of Minnesota, Twin Cities Campus
Jimmy John's
22nd Ave. S
The Cube
East Bank
Village Wok
SE Washington Ave.
122
122
Weisman Art Museum
EAST BANK
Kimchi Tof
SE Delaware St.
Oak Street
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

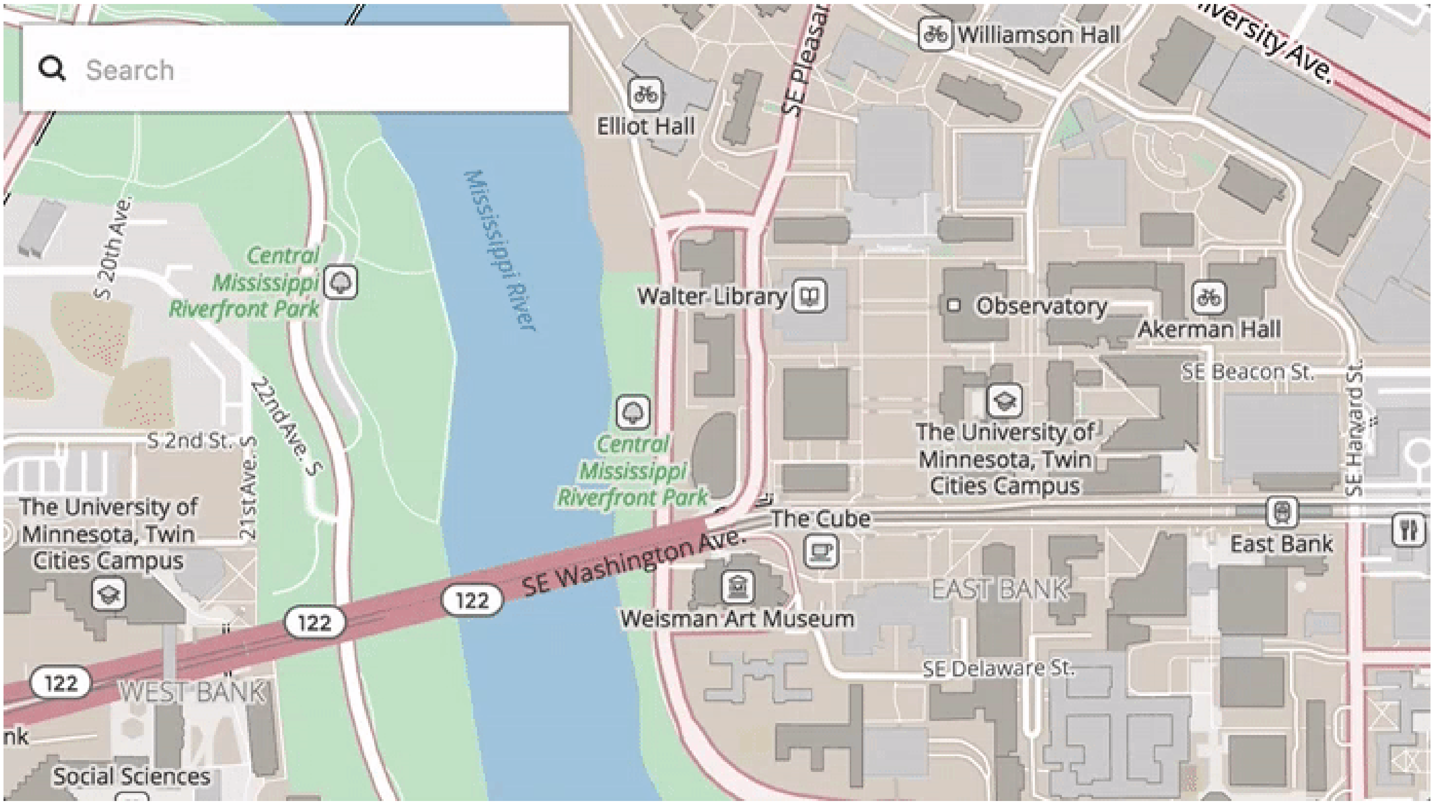
Splendid.

Custom markers

Before we wrap up this example, let’s make one more customization to our map: custom markers.

You may have noticed that many of our search results come with a default Leaflet marker. This marker appears when the result has been highlighted (using the arrow keys on your keyboard) or selected from the dropdown list.

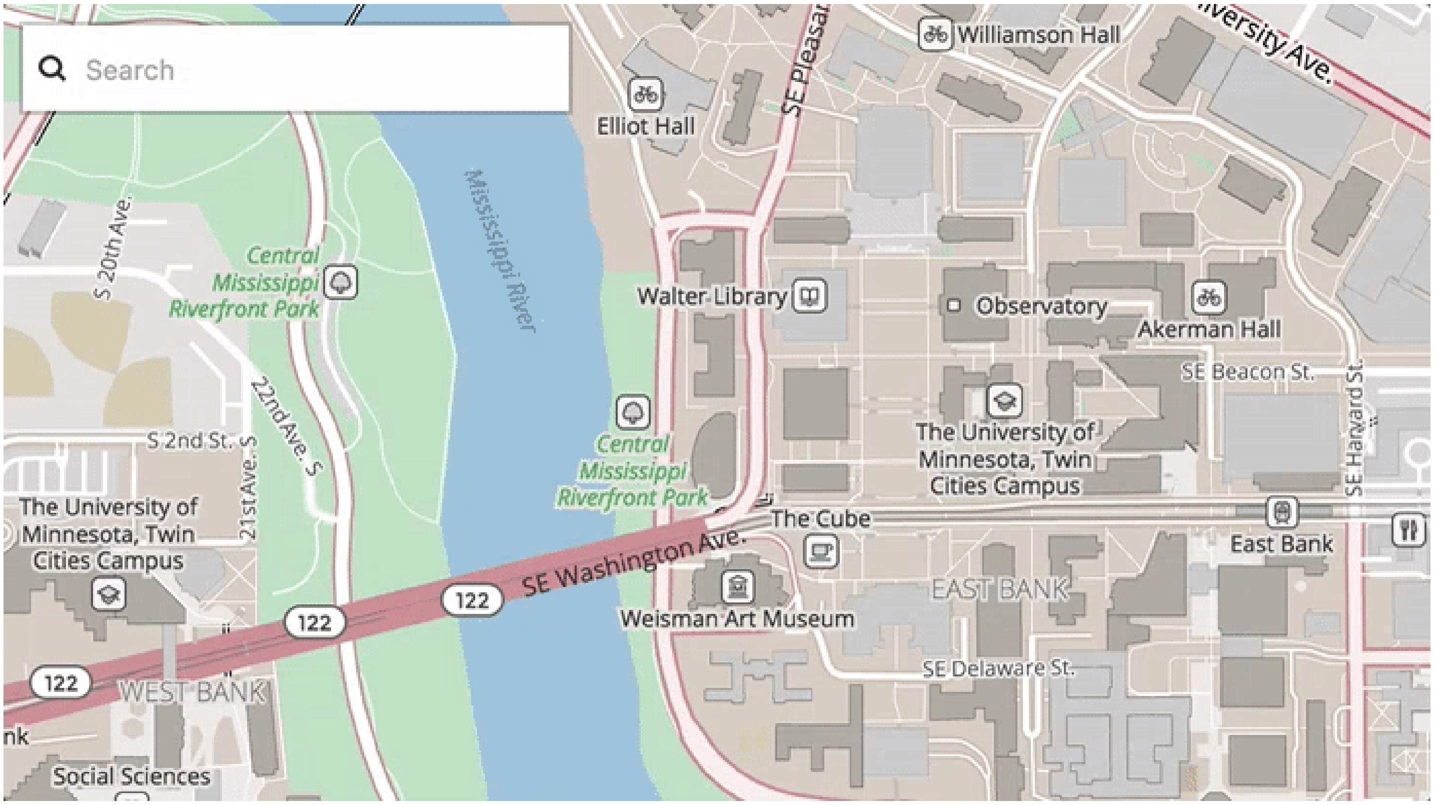
For example:


Search
Williamson Hall
University Ave.
Elliot Hall
SE Pleasant
S 20th Ave.
Central Mississippi Riverfront Park
Mississippi River
Walter Library
Observatory
Akerman Hall
SE Beacon St.
22nd Ave. S
S 2nd St.
21st Ave. S
The University of Minnesota, Twin Cities Campus
Central Mississippi Riverfront Park
SE Harvard St.
The Cube
East Bank
SE Washington Ave.
122
EAST BANK
Weisman Art Museum
SE Delaware St.
WEST BANK
Social Sciences

We can change that icon by using the `markers` option. The `markers` option will accept a boolean (`false` will hide the markers all together) or a set of **Leaflet marker options (http://leafletjs.com/reference.html#marker-options)**, including a custom **L.icon (http://leafletjs.com/reference.html#icon)**.

Let’s set up a custom icon and add that to our geocoder options:

```
// Create custom icon
var myIcon = L.icon({
    iconUrl:  'mn_icon.png',
    iconSize: [40, 40],
    popupAnchor: [0, -20]
});
var markerOptions = {
  icon: myIcon
};

// Define geocoder options
var geocoderOptions = {
  bounds: umnBounds,
  layers: ['venue'],
  markers: markerOptions
};
```


Search
Williamson Hall
University Ave.
Elliot Hall
SE Pleasant
S 20th Ave.
Central Mississippi Riverfront Park
Mississippi River
Walter Library
Observatory
Akerman Hall
SE Beacon St.
SE Harvard St.
22nd Ave. S
S 2nd St.
21st Ave. S
The University of Minnesota, Twin Cities Campus
The Cube
East Bank
SE Washington Ave.
122
Weisman Art Museum
EAST BANK
WEST BANK
SE Delaware St.
Social Sciences

Ok, so maybe my icons are a little on the nose. 😉

Here's a look at our final campus map:

© Mapzen, OpenStreetMap, and others | Leaflet

Country Finder

In the previous example, we focused on a very small area with fine-grained geocoding. For this next example, we'll go in the completely opposite direction. We are going to make an app that searches country names only.

Here's the code to get started with:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Country Finder | The mapzen.js geocoder</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="https://mapzen.com/js/mapzen.css">
    <script src="https://mapzen.com/js/mapzen.min.js"></script>
    <style>
      html,body{margin: 0; padding: 0}
      #map {
        height: 100%;
        width: 100%;
        position: absolute;
      }
      #title {
        position: absolute;
        z-index: 1000;
        background: rgba(255, 255, 255, 0.85);
        border-radius: 5px;
        margin: 10px;
      }
      #country_name {
        color: red;
        font-family: "Helvetica Neue", Helvetica, sans-serif;
        font-weight: normal;
        margin: 10px;
      }
      #initial_text {
        color: gray;
      }
  </style>
  </head>
<body>
  <div id="map"></div>
  <div id="title">
    <h2 id="country_name"><div id="initial_text">Country Finder</div></h2>
  </div>

  <script>
    // Mapzen API key (replace key with your own)
    // To generate your own key, go to https://mapzen.com/developers/
    L.Mapzen.apiKey = 'mapzen-JA21Wes';

    var map = L.Mapzen.map('map', {
      center: [37.632, -96.108],
      zoom: 3
    });
```

```
      // Define geocoder options
      var geocoderOptions = {
        position: 'topright',
        layers: ['country'],
        placeholder: 'Search for a country',
      };

      // Instantiate the geocoder (a.k.a the search box)
      var geocoder = L.Mapzen.geocoder(geocoderOptions);

      // Add geocoder to map
      geocoder.addTo(map);

      // Add results listener

    </script>
  </body>
  </html>
```

This time, we're starting you off with a few `geocoderOptions` already defined.

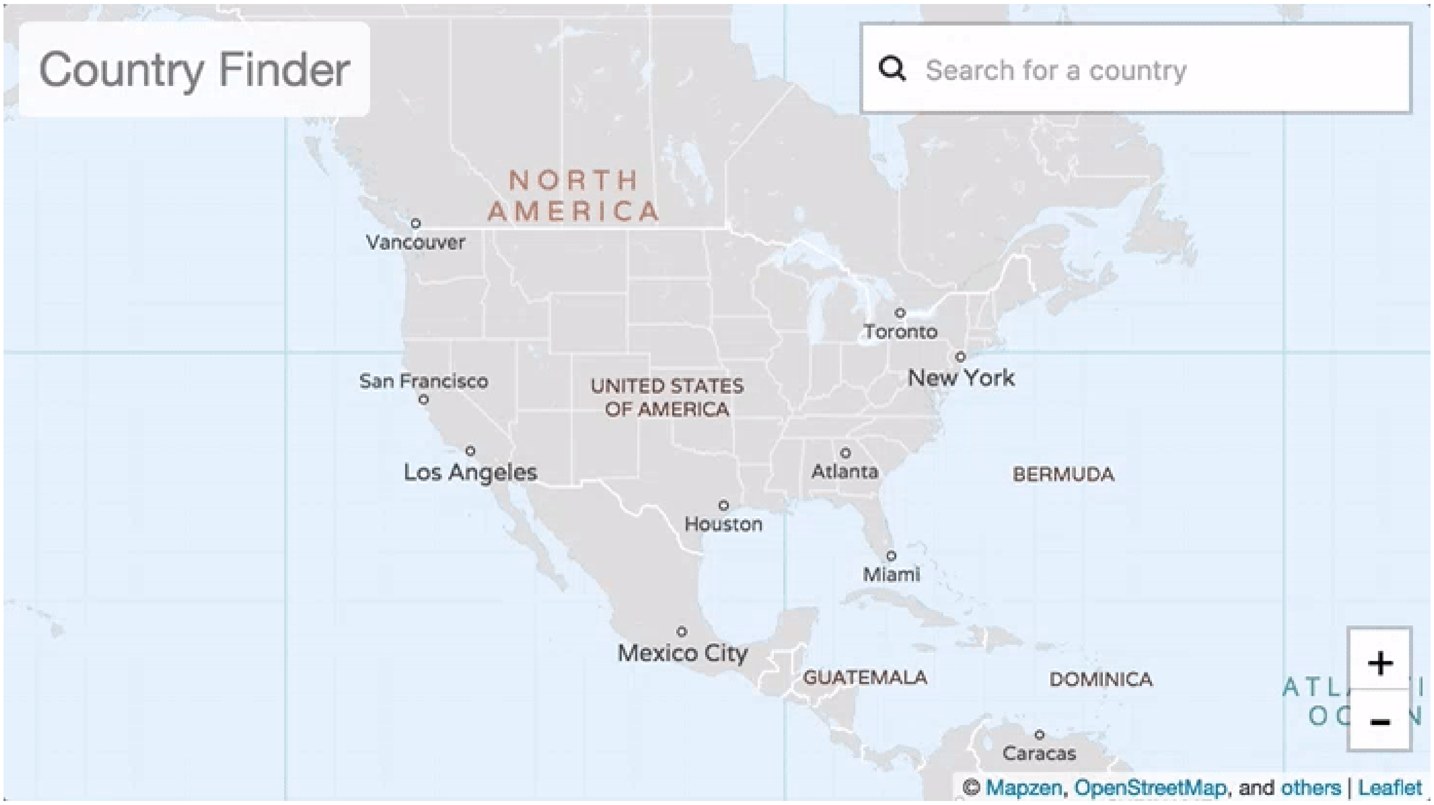
We've added a title to the top left corner of the map, so we moved the geocoder to the `'topright'` of the map. The geocoder (like most Leaflet controls) will take any of **Leaflet's control positions (http://leafletjs.com/reference.html#control-positions)** via the `position` option.

We've also limited the types of places we can search for (using the `layers` option) to simply `'country'`. The geocoder won't return results for any other types. (Again, here's that **full list of available place types (https://mapzen.com/documentation/search/search/#filter-by-data-type)**.)

Lastly, we've updated the placeholder text (that's the text inside of the search box) from the default "Search" to "Search for a country". This should help our users understand that the geocoder will only be returning country results.

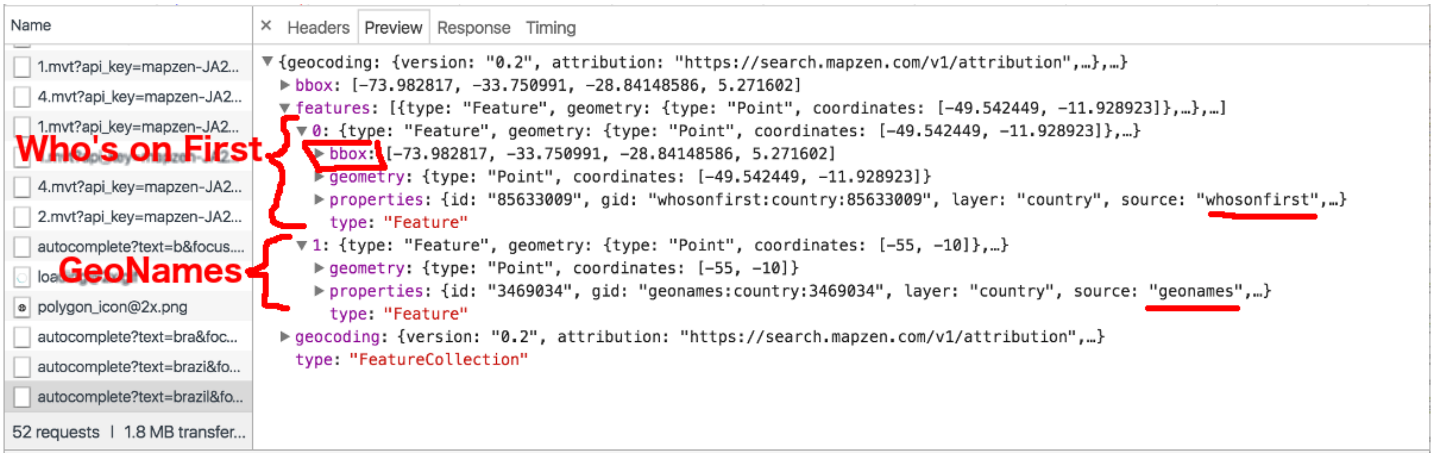
So, even from the start, our map mostly does what we want. You can search for countries and they'll be returned. But there are a few things we can improve upon.

For example, try searching for "Brazil".


Country Finder
Search for a country
NORTH AMERICA
Vancouver
Toronto
New York
San Francisco
UNITED STATES OF AMERICA
Los Angeles
Atlanta
BERMUDA
Houston
Miami
Mexico City
GUATEMALA
DOMINICA
Caracas
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

You'll notice that two results pop up.

If we look at the network request, it appears that the two results are coming from two different sources: whosonfirst and geonames


Name
1.mvt?api_key=mapzen-JA2...
4.mvt?api_key=mapzen-JA2...
1.mvt?api_key=mapzen-JA2...
4.mvt?api_key=mapzen-JA2...
2.mvt?api_key=mapzen-JA2...
autocomplete?text=b&focus....
polygon_icon@2x.png
autocomplete?text=bra&foc...
autocomplete?text=brazi&fo...
autocomplete?text=brazil&fo...
52 requests | 1.8 MB transfer...
×
Headers
Preview
Response
Timing
Who's on First
GeoNames

```
▼{geocoding: {version: "0.2", attribution: "https://search.mapzen.com/v1/attribution",…},…}
  ▶bbox: [-73.982817, -33.750991, -28.84148586, 5.271602]
  ▼features: [{type: "Feature", geometry: {type: "Point", coordinates: [-49.542449, -11.928923]},…},…]
    ▼0: {type: "Feature", geometry: {type: "Point", coordinates: [-49.542449, -11.928923]},…}
      ▶bbox: [-73.982817, -33.750991, -28.84148586, 5.271602]
      ▶geometry: {type: "Point", coordinates: [-49.542449, -11.928923]}
      ▶properties: {id: "85633009", gid: "whosonfirst:country:85633009", layer: "country", source: "whosonfirst",…}
       type: "Feature"
    ▼1: {type: "Feature", geometry: {type: "Point", coordinates: [-55, -10]},…}
      ▶geometry: {type: "Point", coordinates: [-55, -10]}
      ▶properties: {id: "3469034", gid: "geonames:country:3469034", layer: "country", source: "geonames",…}
       type: "Feature"
  ▶geocoding: {version: "0.2", attribution: "https://search.mapzen.com/v1/attribution",…}
   type: "FeatureCollection"
```

The network tab on ***Chrome DevTools (https://developer.chrome.com/devtools)****.*

Sources

Currently, the geocoder searches data from **OpenStreetMap (http://www.openstreetmap.org/)**, **OpenAddresses (https://openaddresses.io/)**, **Who's on First (https://whosonfirst.mapzen.com/)**, and **GeoNames (http://www.geonames.org/)**. Not all sources return the same data (read more about **data sources in Mapzen Search (https://mapzen.com/documentation/search/data-sources/)**).

In this case, both **Who's On First (https://whosonfirst.mapzen.com/)** and **GeoNames (http://www.geonames.org/)** return a (slightly different) result for "Brazil". While Who's on First returns a bounding box for the entire country, the GeoNames result includes information on just the centroid (center-most point) of the country. For our purposes, we will want information about the geography of the entire country–namely, the bounding box.

So, let's limit our data sources to return data just from Who's On First.

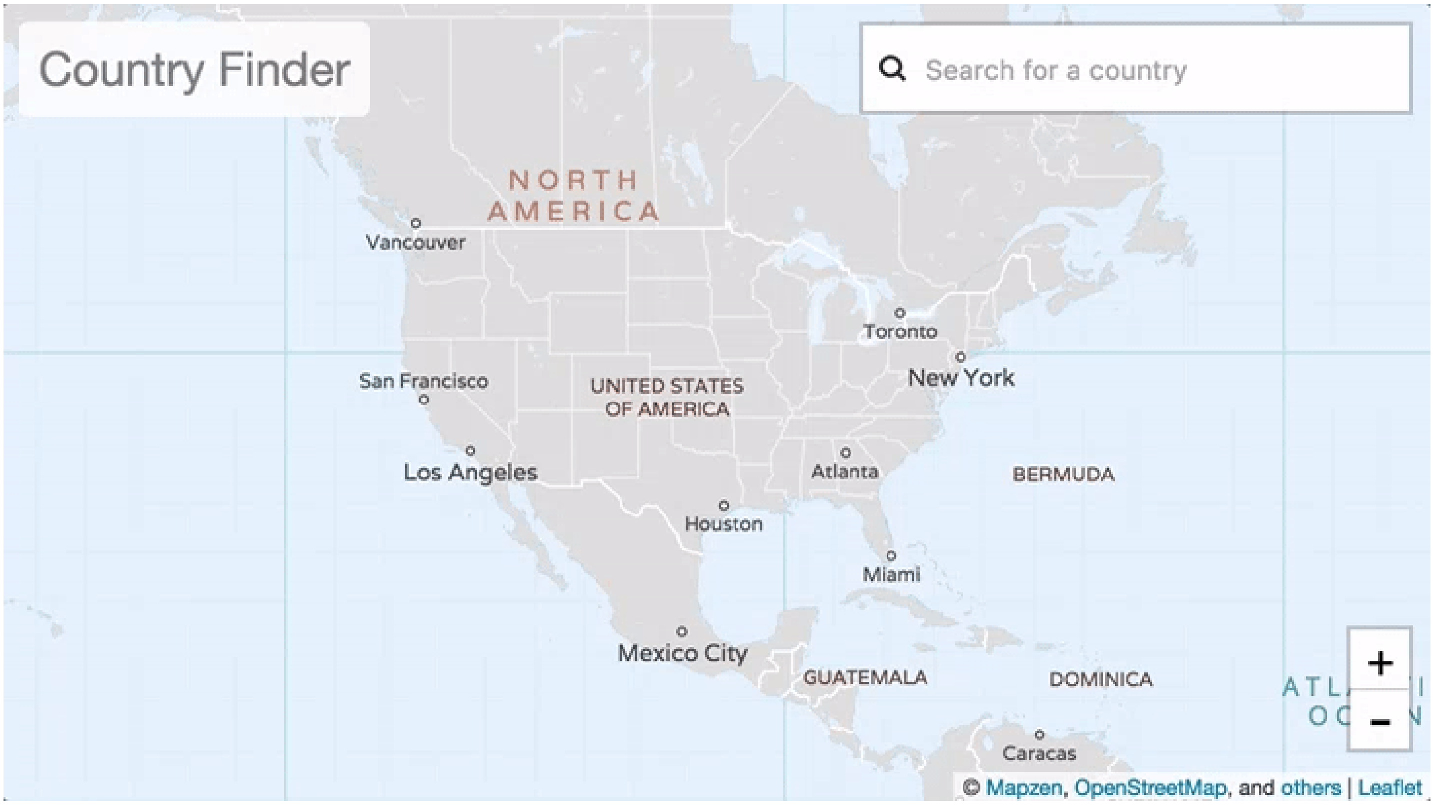
We can do this by using the `params` option. The `params` option will serialize and pass along any **available parameters in Mapzen Search (https://mapzen.com/documentation/search/search/#available-search-parameters)**, including `sources` .

To limit the sources our geocoder will check, add the following to `geocoderOptions` :

```
      // Define geocoder options
      var geocoderOptions = {
        position: 'topright',
        layers: ['country'],
        placeholder: 'Search for a country',
        params: {
          sources: ['wof']
        },
      };
```

(*By the way,* `'wof'` *stands for Who's On First.*)

When we try searching again for "Brazil", we'll see that only one search result is returned.


Country Finder
Search for a country
NORTH AMERICA
Vancouver
Toronto
New York
San Francisco
UNITED STATES OF AMERICA
Los Angeles
Atlanta
BERMUDA
Houston
Miami
Mexico City
GUATEMALA
DOMINICA
Caracas
+
-
© Mapzen, OpenStreetMap, and others | Leaflet

Progress! Now let's do something with that bounding box.

My goal is to highlight the selected country by drawing and zooming to the country's bounding box. We could grab the bounding box from the autocomplete response we just looked at. But autocomplete often returns several results (not to mention, it fires on each keystroke). I'd rather wait until the country has been selected from the list, so we know exactly which country to highlight.

We can do this by leveraging…

The Place Endpoint

Let's jump right in and see this in action. Go ahead and add an option to our `geocoderOptions` called `place` :

```
var geocoderOptions = {
  position: 'topright',
  layers: ['country'],
  placeholder: 'Search for a country',
  params: {
    sources: ['wof']
  },
  place: true,
};
```

By setting this option to `true`, the geocoder will make a request to the **place endpoint (https://mapzen.com/documentation/search/place/)** each time a search result is selected. If we set up an event listener for this request, we can then *do something* with the result.

Let’s add a `results` listener to the bottom of the JavaScript section:

```
// Add results listener
geocoder.on('results', function(e) {
  // requestType may be 'autocomplete', 'search', or 'place'
  if (e.requestType === 'place'){

    // Update country name in title
    var name = e.results.features[0].properties.label;
    document.getElementById('country_name').innerHTML = name;
  }

});
```

So far, all we’re doing is checking the `requestType` (we don’t want this running on every autocomplete response!), updating the country name in our title box, and grabbing the bounding box from the response.

Next, let’s grab the bounding box and convert it into a **Leaflet latlngBounds object (http://leafletjs.com/reference.html#latlngbounds)**. We’ll need this object to highlight and zoom to our selected country.

```
// Add results listener
geocoder.on('results', function(e) {
  // requestType may be 'autocomplete', 'search', or 'place'
  if (e.requestType === 'place'){

    // Update country name in title
    var name = e.results.features[0].properties.label;
    document.getElementById('country_name').innerHTML = name;

    // Get bounding box of result
    var bbox = e.results.bbox;
    var bounds = [
      [ bbox[1], bbox[0] ],
      [ bbox[3], bbox[2] ]
    ];

  }
});
```

To avoid leaving behind a rectangle for each country we've searched, let's create a single **Leaflet Rectangle object (http://leafletjs.com/reference.html#rectangle)** then simply update the bounds of the rectangle each time we select a country.

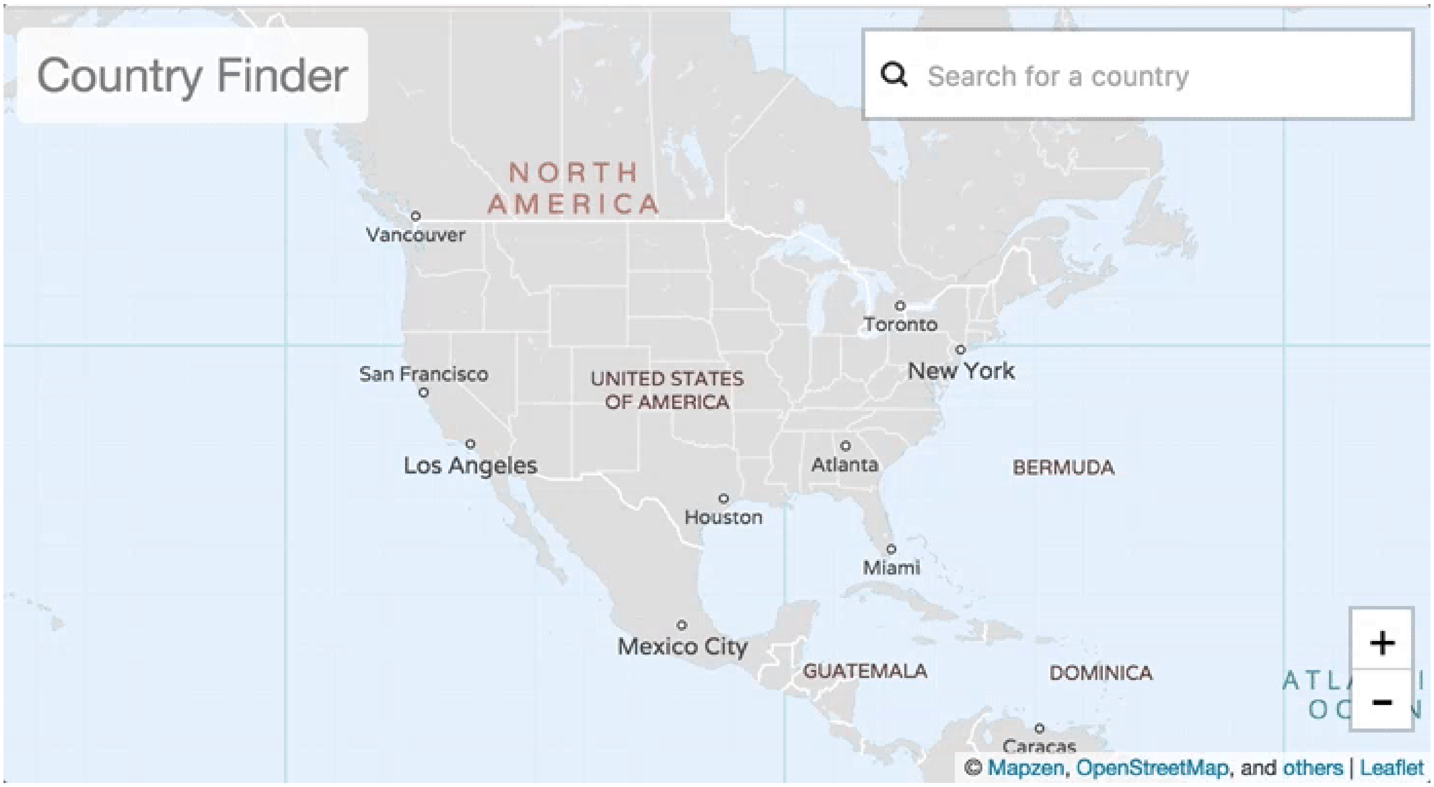
Somewhere near the top of your JavaScript block, set up the initial `L.Rectangle` object:

```
// Set up bounding box for highlighting
var bboxHighlighter = L.rectangle([[180, 90], [180, 90]], {color: 'red', weight: 3, f
```

We'll set the rectangle coordinates to a location outside of our initial map view.

Back in our `results` listener, we'll update the `L.Rectangle` bounds and zoom to the selected country:

```
// Add results listener
geocoder.on('results', function(e) {
  // requestType may be 'autocomplete', 'search', or 'place'
  if (e.requestType === 'place'){

    // Update country name in title
    var name = e.results.features[0].properties.label;
    document.getElementById('country_name').innerHTML = name;

    // Get bounding box of result
    var bbox = e.results.bbox;
    var bounds = [
      [ bbox[1], bbox[0] ],
      [ bbox[3], bbox[2] ]
    ];

    // Update bounds of bounding box
    bboxHighlighter.setBounds(bounds);

    // Zoom to bounding box
    map.fitBounds(bounds);
  }
});
```


Country Finder
Search for a country
NORTH AMERICA
Vancouver
Toronto
San Francisco
New York
UNITED STATES OF AMERICA
Los Angeles
Atlanta
BERMUDA
Houston
Miami
Mexico City
GUATEMALA
DOMINICA
Caracas
ATL
OC
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

Much better.

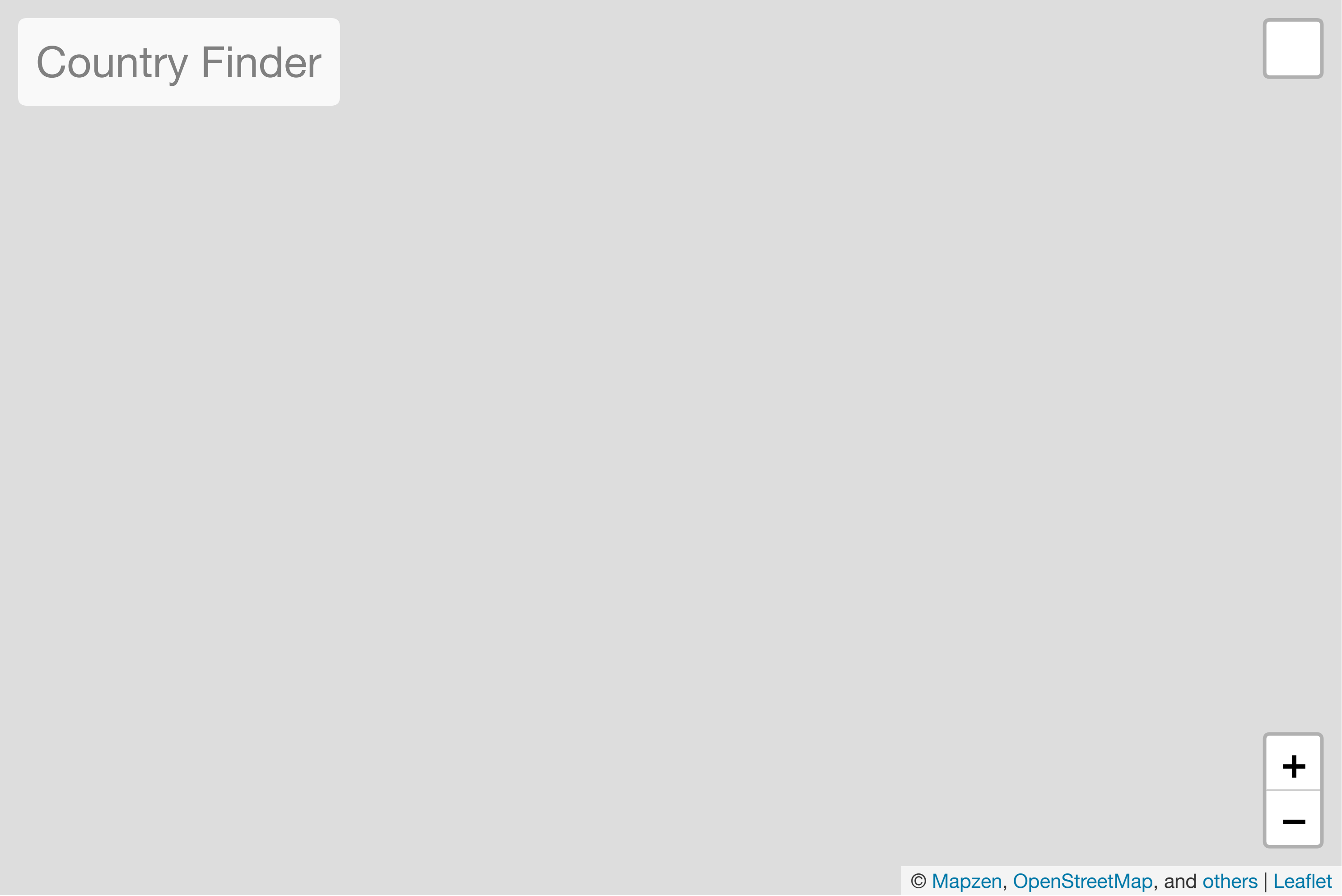
Let’s make two last changes to our Country Finder, just to clean it up a bit.

In our campus map, we added custom markers to the map to show search result locations. For this example, let’s turn off markers altogether by setting the `markers` option to `false`.

While we’re at it, let’s also disable the `panToPoint` feature. This option instructs the map to zoom/pan to a search result as you highlight it in the dropdown list.

```
var geocoderOptions = {
  position: 'topright',
  layers: ['country'],
  placeholder: 'Search for a country',
  params: {
    sources: ['wof']
  },
  place: true,
  markers: false,
  panToPoint: false
};
```

And we’re done!



Phew. Is it just me or are these **Make Your Own (https://mapzen.com/tag/make-your-own/)** posts getting longer? #SorryNotSorry

The full code for these demos can be seen here: **Campus Map (https://bl.ocks.org/rfriberg/8cad1da5499067d68c08c7c505e6bd99)** and **Country Finder (https://bl.ocks.org/rfriberg/6214dedce72ddb03e0fb81d9312825ca)**.

If you have questions or want to show off something you've made with Mapzen, **drop us a line (mailto:hello@mapzen.com)**. We love to hear from you!

~~~

Check out additional tutorials from the **Make Your Own (https://mapzen.com/tag/make-your-own/)** series:

- **One Minute Map (https://mapzen.com/blog/one-minute-map/)**
- **Map Sandwich (https://mapzen.com/blog/map-sandwich/)**
- **Filters & Functions (https://mapzen.com/blog/filters-and-functions/)**
~~~

- **Put A Label On It (https://mapzen.com/blog/tangram-labels/)**
- **Interactive Mapping with Tangram (https://mapzen.com/blog/tangram-interactivity/)**
- **Lots of Dots (https://mapzen.com/blog/lots-of-dots/)**
- **Customize Your Search (https://mapzen.com/blog/customize-your-search/)**

Preview image of binocular telescope via ***Pexels (https://www.pexels.com/photo/binocular-telescope-view-zoom-19593/)*** *(CC0)*

· 14 February 2017 ·

Rhonda Friberg

Rhonda is a javascripter who makes maps, trouble, and the occasional pie.

© 2017 Mapzen

Announcement: API keys required to use Mapzen services

vector-tiles (/tag/vector-tiles) support (/tag/support)

After March 1, 2017, all access to the **Mapzen Tile service (https://mapzen.com/documentation/vector-tiles/)** will require an API key.

Be sure to update any apps that may be sending keyless requests. Any requests for vector or terrain tiles sent without an API key will soon return `401` and `429` errors.

Mapzen provides a **generous free tier of API usage (https://mapzen.com/documentation/overview/#rate-limits)** when you register for a **Mapzen developer account (https://mapzen.com/developers)**. Sign up today!

See examples below on how to add an API key to Tangram and mapzen.js.

API keys are required

Mapzen's mapping APIs enable developers to integrate maps, search, mobility, and place data into web, mobile, and desktop applications. When you register for a **Mapzen developer account (https://mapzen.com/developers)** you are able to access our services with an API key, a code that uniquely links API usage inside your application with your account without needing a password.

Mapzen's web services are resources shared by all our users. We use rate-limiting through API keys to ensure you get the maximum use of the generous limits associated with your account, and to help prevent individual users from degrading system performance for everyone.

In the past, it has been possible to access Mapzen's services without an API key, but at significantly reduced rate limits compared to the levels available when passing an API key. We expected this keyless usage ability to be used only for trying out our services and training purposes, such as in code samples in technical documentation. However, some developers have relied on keyless access in production apps, and this has caused heavy server loads and poor experiences.

We have identified many cases in which end-users of apps that integrate Mapzen's services have experienced broken maps and missing data after exceeding keyless access limits. Because the map works properly when tested, it is typical that developers are unaware that a problem exists, unless their users make the effort to report it.

If you use an API key, however, you can monitor the health of your application and see exactly what your users see.

Because all users should be able to enjoy speed, reliability, and good user experiences, you must include an API key with your request to Mapzen services. **This is enforced March 1, 2017 for the Mapzen Vector Tile and Terrain Tile services.** After that, the server will return error messages, such as HTTP status codes of `401` and `429` , when you attempt to use our APIs without a key. These error messages help you identify issues sooner so you can correct them before they reach your users.

Note that this is in addition to the previous announcement that **Mapzen basemap styles require an API key (https://mapzen.com/blog/basemap-style-update/)** in this same timeframe.

API keys will be enforced for other Mapzen services, such as Mapzen Search and Mapzen Mobility, in the future; watch the **Mapzen blog (https://mapzen.com/blog/)** for dates. In the meantime, start preparing by updating your applications to use an API key.

What you need to do

There are two things you need to do:

1. Sign up for a free API key at **https://mapzen.com/developers (https://mapzen.com/developers)**.
2. Use it in all your requests to Mapzen's APIs.

If you are already following both of these, then nothing changes for you. You do not need to do anything further.

If not, you must take action and integrate Mapzen API keys into your apps. The simplest and most common way to include an API key is for your app to employ a single key with enough capacity to cover all of your users' usage. **Contact us (mailto:hello@mapzen.com)** if you need rate limits above the **default levels (https://mapzen.com/documentation/overview/#rate-limits)**.

If your app currently provides an option for users to enter their own Mapzen API keys, you should indicate that this setting is required, or redesign your app to follow the centralized key model.

Examples

Below are examples of how to add an API key in Tangram YAML, Tangram JS and Mapzen.js:

Example **Tangram YAML** usage:

```
import: https://mapzen.com/carto/refill-style/6/refill-style.yaml

global:
  sdk_mapzen_api_key: mapzen-xxxxxx
```

Example **Tangram JS** usage:

```
var layer = Tangram.leafletLayer({
  scene: {
    import: 'https://mapzen.com/carto/refill-style/6/refill-style.yaml',
    global: { sdk_mapzen_api_key: 'mapzen-xxxxxx' }
  });
```

Example **Mapzen.js** usage:

```
<script>
  L.Mapzen.apiKey = 'mapzen-xxxxxx';

  var map = L.Mapzen.map('map', {
    center: [40.8041, -124.1506],
    zoom: 15,
    maxZoom: 20,
    tangramOptions: {
      scene: L.Mapzen.BasemapStyles.Refill
    }
  });
</script>
```

See the **Mapzen developer overview (https://mapzen.com/documentation/overview/)** for more information on how to create and use API keys.

API keys for your testing and training needs

We are working on a system to support the original keyless access use cases of demos and testing. Future enhancements will allow you to copy code examples from technical documentation and paste them directly into your projects – check the **Mapzen blog (https://mapzen.com/blog)** for updates.

Preview image ***via Cooper-Hewitt (https://collection.cooperhewitt.org/objects/18340181/)****: 19th century glazed pottery, Museum purchase through gift of Mary Hearn Greims.*

· 22 February 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

Street address interpolation

search (/tag/search) geocoding (/tag/geocoding) data (/tag/data)

engineering (/tag/engineering)

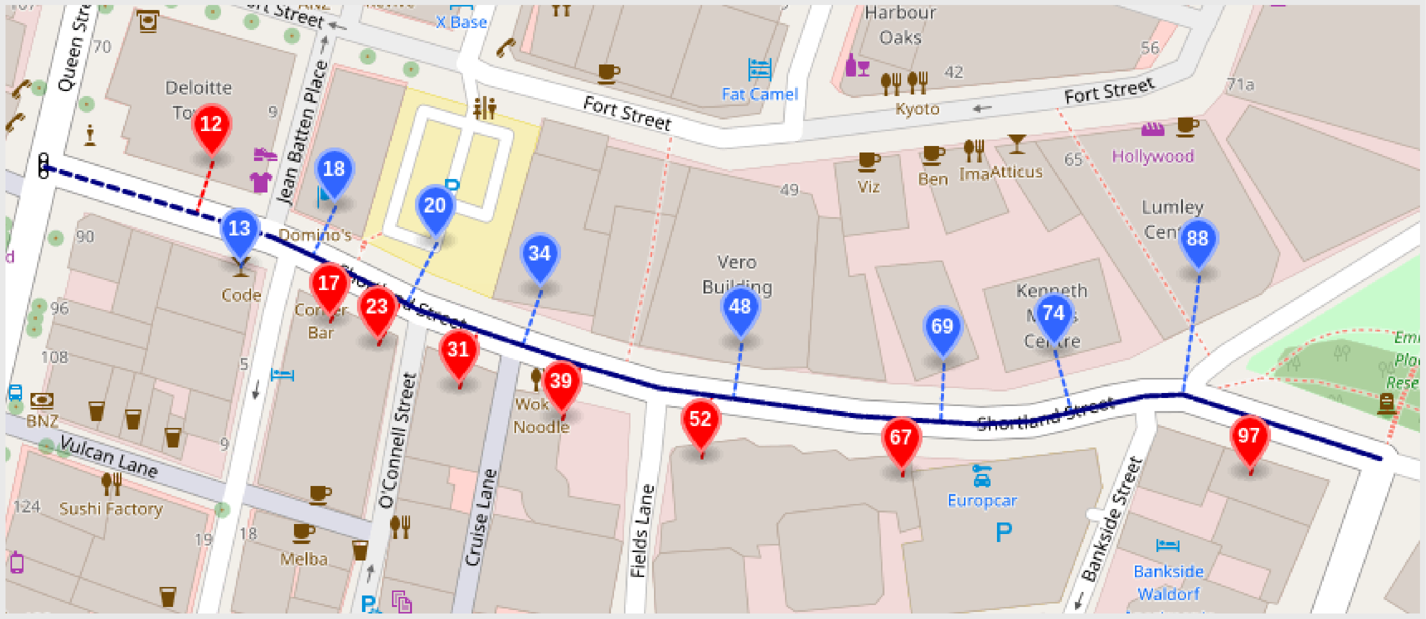
Today we’re announcing a new address interpolation system. It was designed for the **Pelias geocoder (https://github.com/pelias/pelias)** and is now available in **Mapzen Search (https://mapzen.com/products/search/)**, but can also be used as a stand-alone application or be included with any other geographic software or search engine.

The term ‘interpolation’ refers to a method of creating new data points from a set of known data points. For geographic search, this is important for street address data where some house numbers may be missing from our data providers.

The **OpenStreetMap (http://www.openstreetmap.com)** and **OpenAddresses (http://www.openaddresses.com)** projects provide a huge cache of street address information; between them over 500 million geographic address points are freely available to download and use for routing people and cars from A to B.

The issue is that these datasets are rarely complete, they contain 'holes’ in the data where house numbers or geographic coordinates are not available.

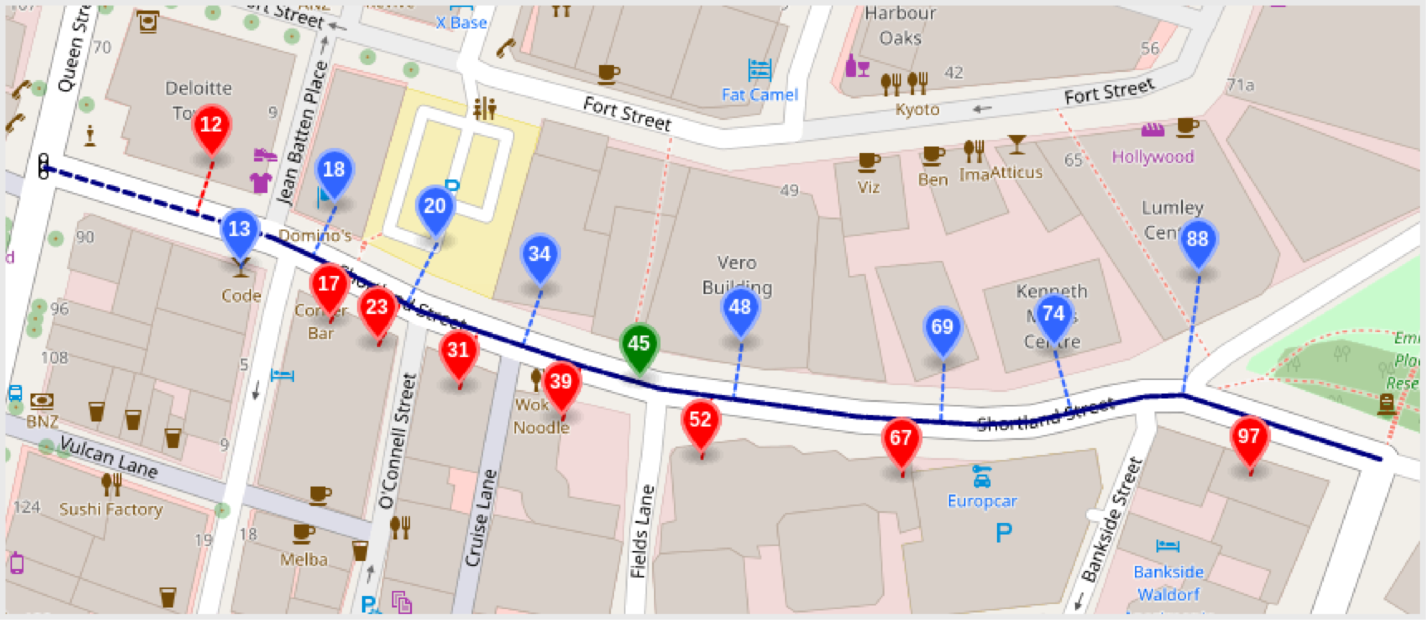
> Consider a driver asking for directions to `45 Shortland Street, Auckland, NZ` :


Queen Street
Deloitte
Jean Batten Place
Fort Street
Fort Street
X Base
Fat Camel
Harbour
Oaks
Kyoto
Viz
Ben
ImaAtticus
Hollywood
Lumley
Vero
Building
Kenneth
Domino's
Code
Bar
Wok
Noodle
Shortland Street
O'Connell Street
Cruise Lane
Fields Lane
Bankside Street
Europcar
Bankside
Waldorf
Vulcan Lane
Sushi Factory
Melba
BNZ
12
18
13
20
17
23
31
34
39
48
52
67
69
74
88
97

Red and blue dots are known address points from OpenStreetMap and OpenAddresses.

The 'holes' in the data result in our search engine returning a less granular result for queries which request the location of one of these missing addresses.

By filling in these gaps in the data, we can intelligently estimate where the missing house numbers would lie on each road:


Queen Street
Deloitte
Jean Batten Place
Fort Street
Fort Street
X Base
Fat Camel
Harbour
Oaks
Kyoto
Viz
Ben
ImaAtticus
Hollywood
Lumley
Vero
Building
Kenneth
Domino's
Code
Bar
Wok
Noodle
Shortland Street
O'Connell Street
Cruise Lane
Fields Lane
Bankside Street
Europcar
Bankside
Waldorf
Vulcan Lane
Sushi Factory
Melba
BNZ
12
18
13
20
17
23
31
34
45
39
48
52
67
69
74
88
97

The system is also aware of the geometric shape of each street and uses this information, along with the point data, to ensure that the location it returns lies on the road network and not somewhere which could be a potential driving hazard.

Interpolated data points are never as precise as the rooftop accuracy data we get from our providers. However, we aim to provide a location which is not only safe to drive to but also close enough for a driver to visually identify the letter box or driveway of their destination.

Improving address discovery

Over the last few months we have been rolling out improvements to Mapzen Search on indexing and retrieving postal addresses.

Back in September we finished a project which imports all of the **OpenStreetMap road network in to Pelias (https://github.com/pelias/polylines)**. This first step allowed us to return the mid-point of each road geometry in lieu of having the coordinates for an exact house number.

The next step was to produce an interpolation engine which could go one step further and return an interpolated address location based off the other data points we have for that street.

Design goals

We set out with the following strategic goals in mind:

- Ensuring every street in OpenStreetMap is indexed and retrievable.
- Supporting address ranges as provided by OpenStreetMap, TIGER, et al.
- Combining and de-duplicating distinct address point data sets.
- Designing the system to scale beyond 1 billion address points.
- Allow room for future extension / improvements.

These changes will improve the user experience by:

- Providing house number interpolation where address range data exists.
- Falling back to a street centroid in lieu of a satisfactory house number.
- Reducing noise by only showing a maximum of one result per street.

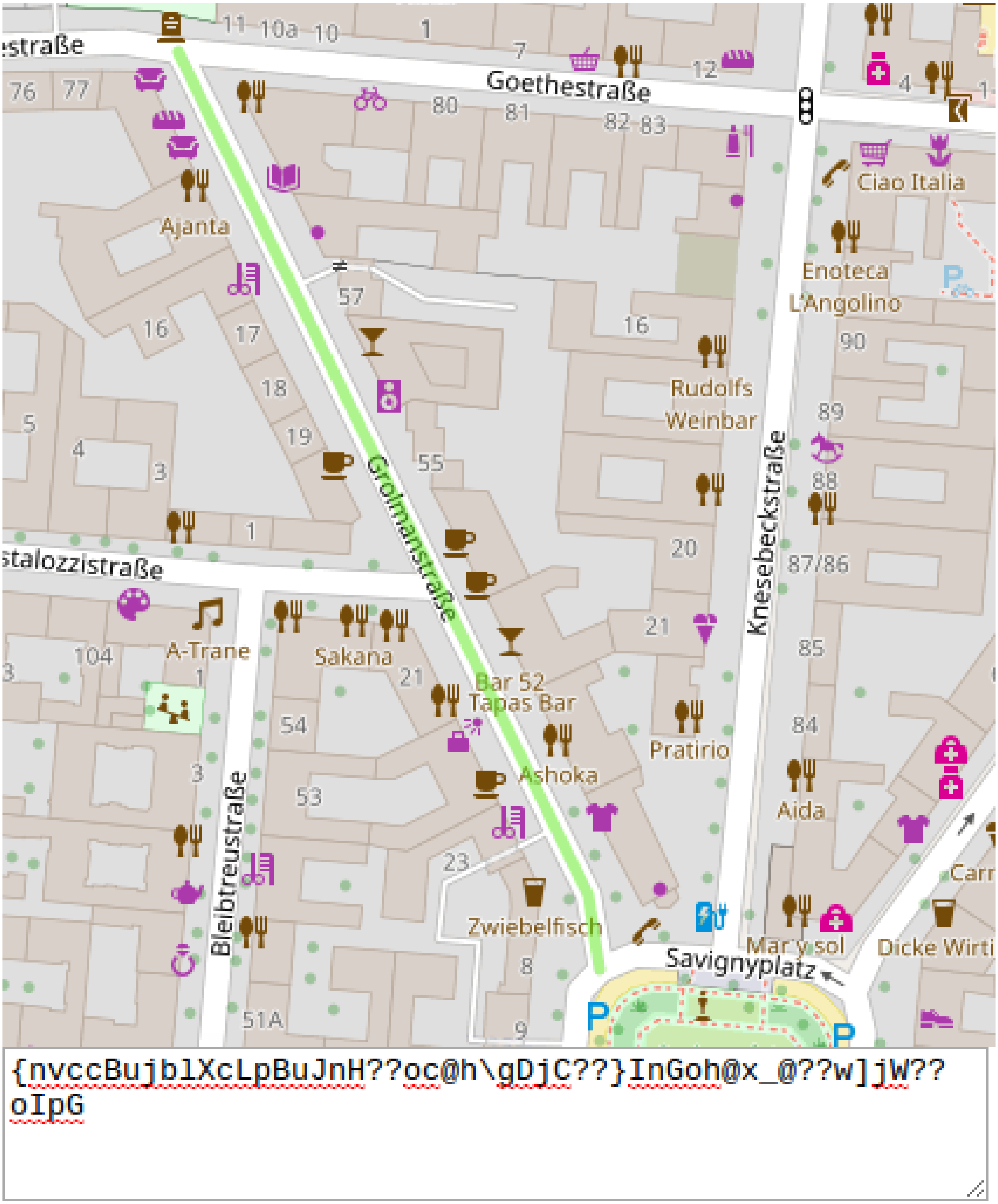
How it works

Road network

First we need a good source of open road geometry data. We can get this from OpenStreetMap.

The data is provided as 'ways' which first need to be joined together in order to form the longest contiguous line string. This will allow us to not only interpolate over 'ranges' of data (such as blocks) but also along the whole length of the street, even if it runs between different countries!

To accomplish this we use the **Valhalla (https://mapzen.com/blog/valhalla-intro/)** routing engine to export the routing graph into an encoded line geometry format called **Polyline (https://developers.google.com/maps/documentation/utilities/polylinealgorithm)**.

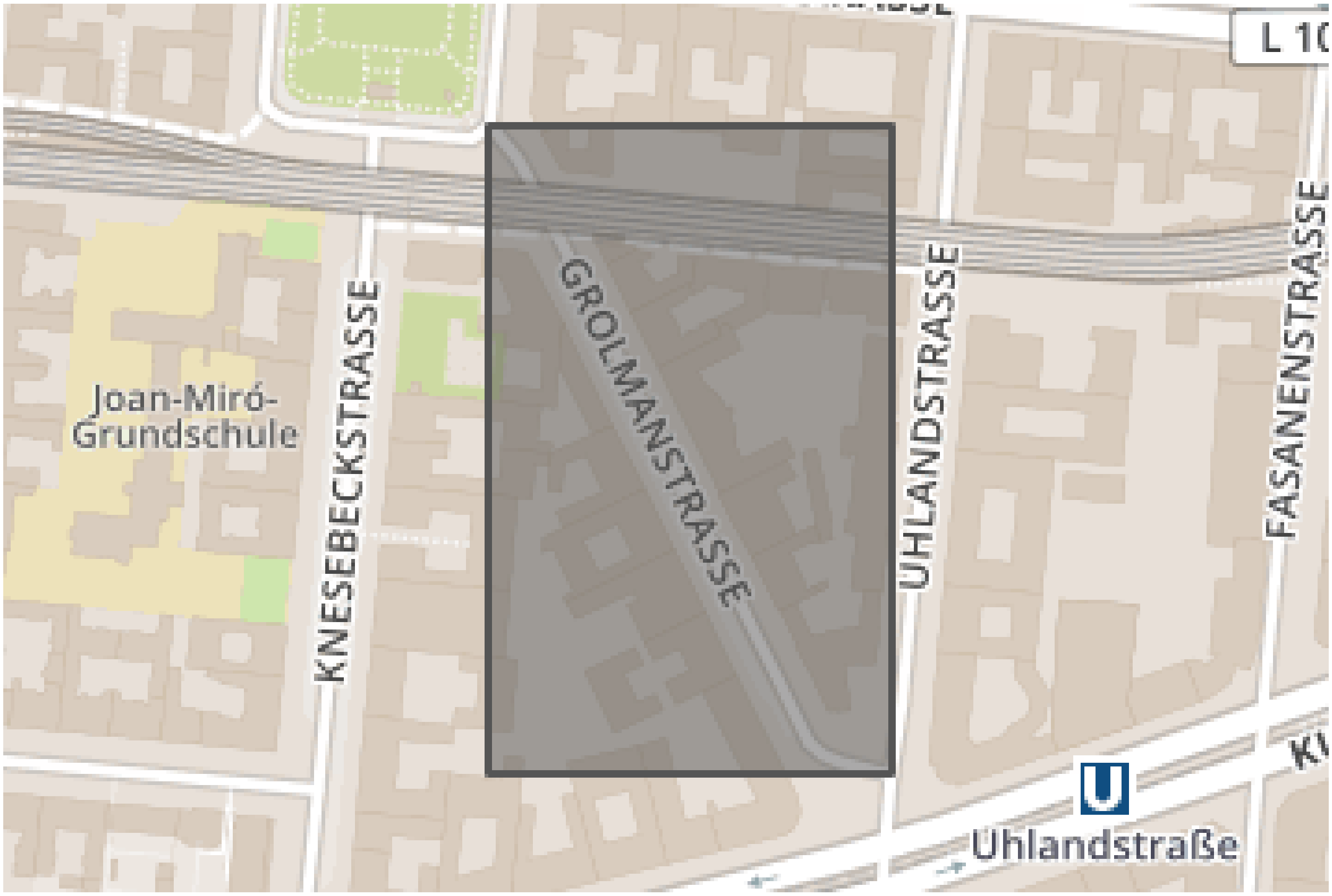

Goethestraße
Grolmanstraße
Knesebeckstraße
Bleibtreustraße
Savignyplatz
Ajanta
A-Trane
Sakana
Bar 52
Tapas Bar
Ashoka
Zwiebelfisch
Rudolfs
Weinbar
Pratirio
Aida
Ciao Italia
Enoteca
L'Angolino
Mar y sol
Dicke Wirti

```
{nvccBujblXcLpBuJnH??oc@h\gDjC??}InGoh@x_@??w]jW??
oIpG
```

The polyline for Berlin's Grolmanstraße

Once we have these polylines for each street in the world, we can go ahead and compute a point halfway along the path to use as a mid-point value, then those streets can be imported directly in to **Pelias (https://github.com/pelias/polylines)** for use in a 'fallback' scenario when we cannot find the exact address the user is searching for.

For the purposes of interpolation, we use the same polyline dump to import those geometries in to `sqlite3` . We use the **rtree (https://www.sqlite.org/rtree.html)** module to create a fast spatial index of the street bounds, then we store the geometry along with the bounds and the street name(s) in a database called `street.db` .



Putting Grolmanstraße in a bounding box.

In order to normalize the street names we run the street names through the **libpostal (https://github.com/openvenues/libpostal)** address expander. This allows us to do things like remove diacritics, handle unicode and perform synonym substitutions.

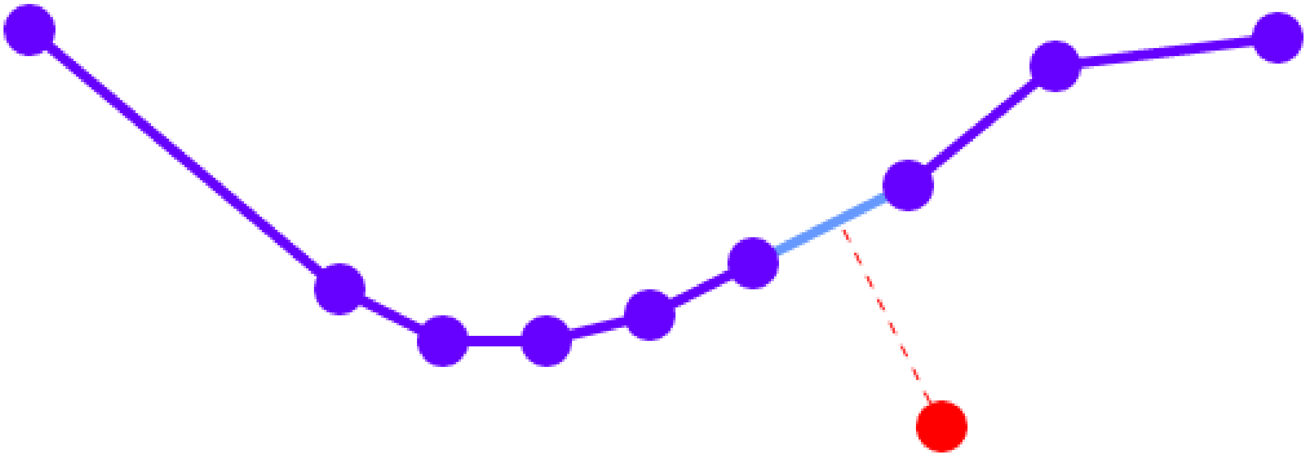
The importer takes a few hours to run and produces a single `street.db` file (~5.3G) which can be shared with any other applications which need a conflated road network data set. Since the file is `sqlite3` it can be accessed from any programming language or copied to a embedded device such as a mobile phone or even your car!

Addresses Points

Next we download >400 million **OpenAddresses (https://openaddresses.io/)** (OA) and perform street name normalization which allows us to use text matching techniques to match them with the road network.

For each data point in the OA files we attempt to find a corresponding street in the `street.db` using the rtree index. This results in a >80% success rate for matching one data set with the other.

Now we have the address points and the street geometry we can then 'project' each point on to the shape of the street. To do this we use a linear algebra function to find its orthogonal projection; this is basically the point on the road closest to the front door of the house.



Click for an **interactive demo (http://geo.wiz.co.nz/projection.html)**.

We then store each address in a database called `address.db` : each row contains both the original position and the projected position, along with id of the corresponding street in `street.db` .

But it's not only OpenAddresses! We also import OpenStreetMap entities tagged with both `addr:street` and `addr:housenumber` .

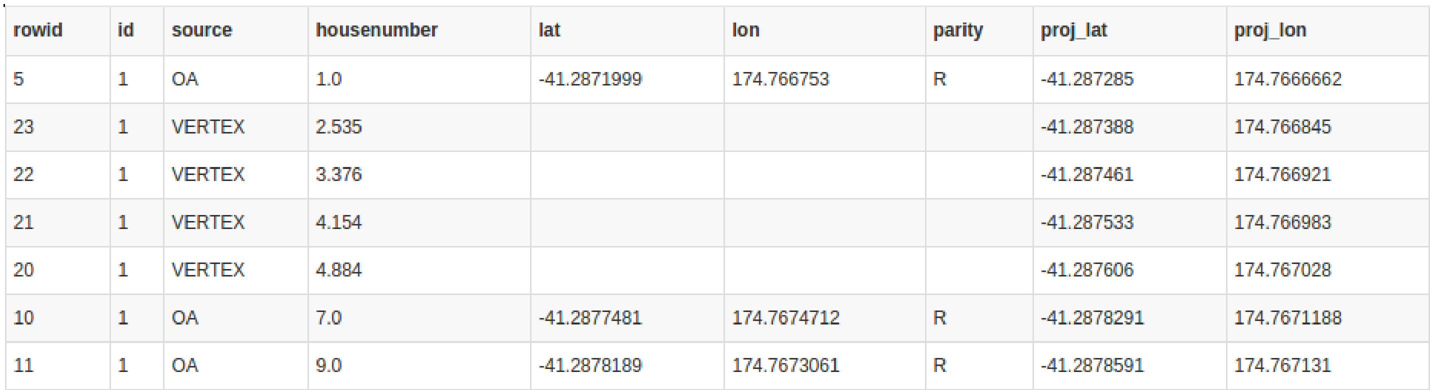
We then finish off by importing all the block ranges from the **US Census Dept 2016 TIGER files (https://www.census.gov/geo/maps-data/data/tiger.html)**, giving us a very high level of coverage in the United States.

Following the contour of the street

The final part of the build is to perform some pre-computation on the street geometry itself; we want to build an index which is highly performant at search time.

In order to avoid loading the street geometry every time we want to perform a search, we iterate over each vertex (corner) of the street geometry and compute a 'fractional house number’.

Using the data we already have about where each house number lies on the path, we can now ask the system “if this corner of the road was a house, what house number would it have?”



rowid	id	source	housenumber	lat	lon	parity	proj_lat	proj_lon
5	1	OA	1.0	-41.2871999	174.766753	R	-41.287285	174.7666662
23	1	VERTEX	2.535				-41.287388	174.766845
22	1	VERTEX	3.376				-41.287461	174.766921
21	1	VERTEX	4.154				-41.287533	174.766983
20	1	VERTEX	4.884				-41.287606	174.767028
10	1	OA	7.0	-41.2877481	174.7674712	R	-41.2878291	174.7671188
11	1	OA	9.0	-41.2878189	174.7673061	R	-41.2878591	174.767131

This step is very important because it allows us to preserve the shape of the street, meaning we will never return a point which does not lie on the path.

We designed the system in such a way that a vehicle driver would never be given a destination which lies in a hazard or is not routable using turn-by-turn navigation systems.

Contributing & Feedback

As always, this project is open-source and **available on Github (https://github.com/pelias/interpolation)**. Please support the project by opening descriptive issues when you find them. (Note that when Mapzen Search returns an interpolated address, you’ll see `interpolated` for the `match_type` in the results.)

```
  },
  "properties": {
    [...]
    "name": "207 Spear Street",
    "housenumber": "207",
    "street": "Spear Street",
    "confidence": 0.8,
    "match_type": "interpolated",
    "accuracy": "point",
    [...]
  }
```

We will be soon be offering database downloads of the interpolation index and we'll continue to improve the data quality and interpolation algorithms as we work towards having over a billion address points available for search worldwide.

As always, we would love to get your feedback on Mapzen Search and our open source geocoding engine, Pelias. You can find us on **here (http://github.com/pelias/pelias)**, **there (https://gitter.im/pelias/pelias)**, and **way over there (search@mapzen.com)**

· 23 February 2017 ·

Peter Johnson

Peter works on the search team, helping people get from A to B.

© 2017 Mapzen

Open Traffic technical preview #1: OSMLR segments

Since we announced our collaboration with the World Bank and more partners to create the Open Traffic platform (/blog/announcing-open-traffic), we've been busy. We're now ready to share a technical preview of one of the key components of the platform: the OSMLR linear referencing system.

At its core, traffic data is all about current or historical speeds along a section of road. Specifying speed is easy, once a choice of units has been made, but specifying a section of road is more complex. There are many reasons for this:

- We are using data from **OpenStreetMap (https://www.openstreetmap.org)**, which provides a detailed, global road network with the same position on openness and accessibility that Open Traffic aspires to. However, the OpenStreetMap (abbreviated to OSM) data is constantly updated as myriad people around the world contribute to it every minute of the day. While it would be possible for Open Traffic to follow the same approach, we felt that there were benefits to having a slower, more controlled update process.
- Although we are using data from OSM, many individuals and fleet operators who wish to contribute to the project may want to use their own local datasets, and we want to make it possible for them to be part of Open Traffic. This means that, although we construct our references to road sections from OSM data, it should be possible to find the section of road on a different map.
- Different datasets have different methods of splitting up roads into smaller parts, often to satisfy different requirements about their data model. For specifying traffic data, there is a balance between making road segments as long as possible to decrease the space that traffic data takes up, and making road segments shorter to increase the level of detail in the data. Getting this right for Open Traffic will require a different method of splitting up roads than any other dataset has chosen.
- To keep data usage to a minimum, which may be important for sending traffic data to mobile devices, we want OSMLR descriptors and identifiers to be compact.

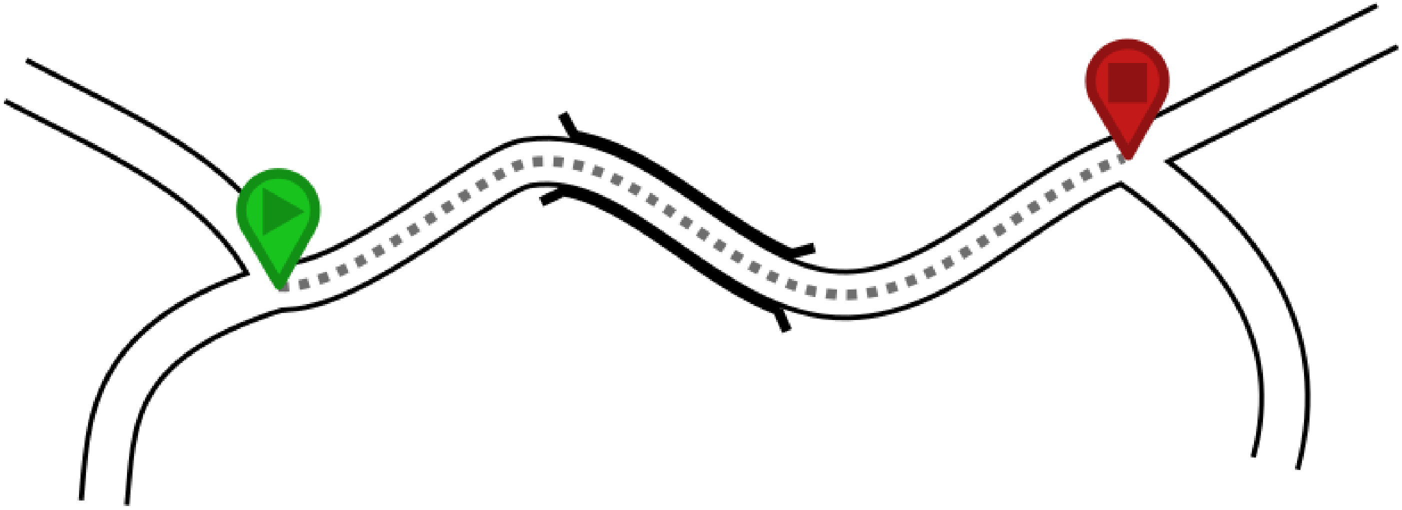
How does OSMLR solve the problem?

To specify a section of road, OSMLR (OpenStreetMap Linear Referencing) describes it as a series of “Location Reference Points” (LRPs) plus a final coordinate. Each LRP contains information about the road at that point and along to the next LRP, which help to uniquely identify the road section. The LRPs can be quite widely spaced as long as the route between two LRPs is uniquely identified by the endpoints and the properties attached to the LRP.

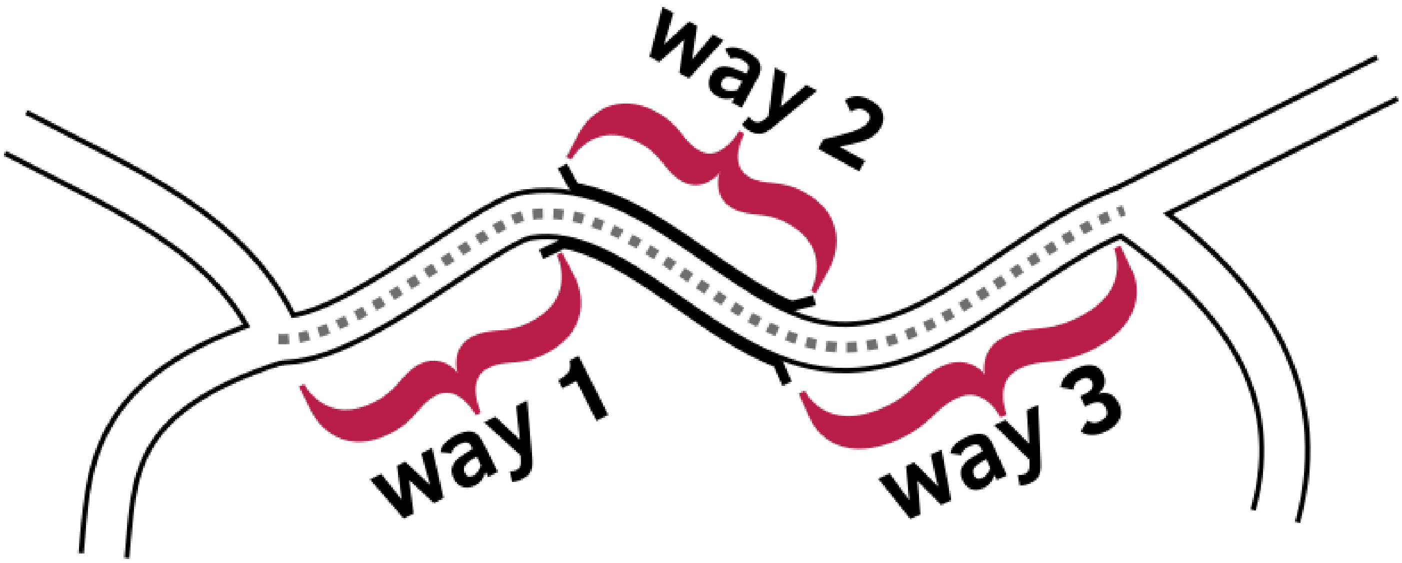
Those properties are:

- Bearing along the road from the LRP. Specifically, this is taken as the bearing from the LRP to a point a short distance further along the road from the LRP in the direction of the next coordinate. This helps to find the right section of road at junctions where many sections may start.
- Road class at the LRP. This helps to identify which of several roads meeting at a point, or flying over or under each other, is intended by the descriptor.
- “Form of Way” at the LRP. This also helps to identify the particular road which is referred to from the potentially many which might be found at a given point.
- Least road class between the LRP and the next coordinate. This tries to reduce ambiguity if multiple paths exist between the LRP and the next coordinate. For example, if the shortest path between the LRP and the next coordinate takes a detour away from the highway, but the least road class is set to highway, then we know that we must adjust the path.
- Length of the road between the LRP and the next coordinate. This avoids matching long detours if roads are missing in the dataset being matched to.

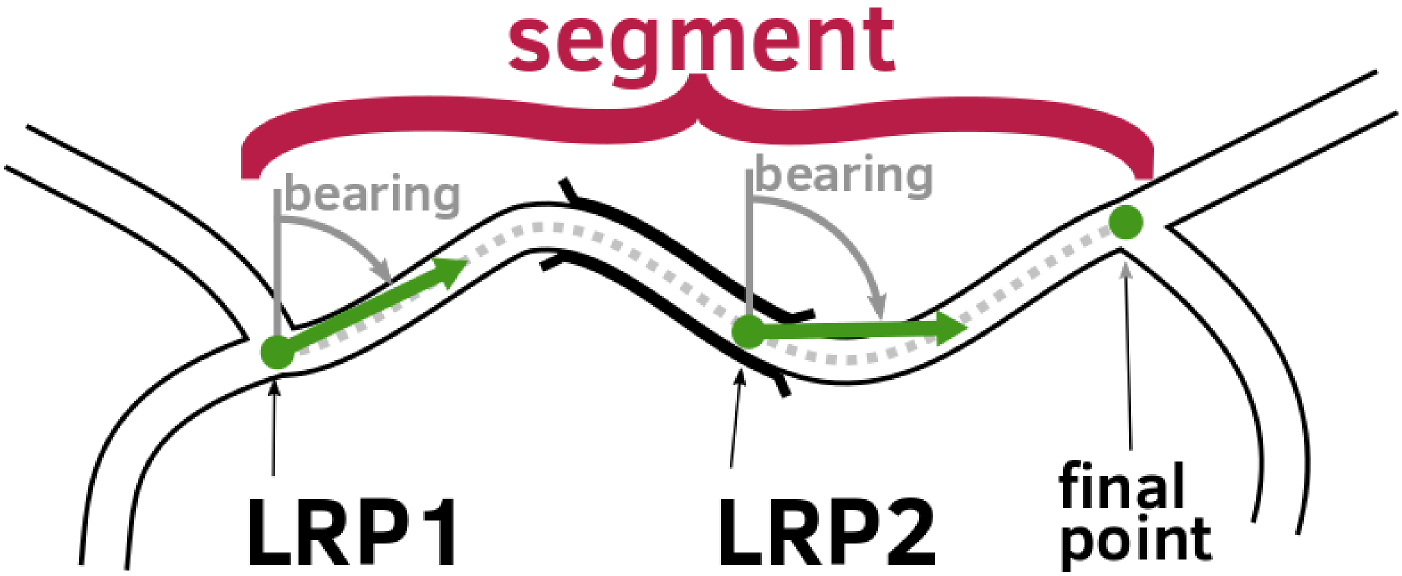
To illustrate, consider the following fictional section of road.



In this example, we want to turn the contiguous section of road between the two junctions, which includes a section of bridge, into an OSMLR segment. However, because the bit of road which is a bridge has different properties, this means that there will be (at least) three OSM ways.



We need to merge across these ways to make up the whole segment. Once that segment has been identified, we need to compute the descriptor, which is the series of LRPs that describe it.



The LRPs are not aligned with the ways, and do not make reference to them. For each LRP, the properties are calculated along the geometry of the combined segment. In this example, there are two LRPs, which is not really to scale as LRPs only need to be repeated every 15km.

The whole descriptor might be:

1. First LRP coordinates:
    - Bearing: `60 degrees`
    - Road class: `Tertiary`

   - Form of way: `Single carriageway`
   - Least road class: `Tertiary`
   - Length: `50m`
2. Second LRP coordinates:
   - Bearing: `90 degrees`
   - Road class: `Tertiary`
   - Form of way: `Single carriageway`
   - Least road class: `Tertiary`
   - Length: `40m`
3. Final point coordinates (with no properties).

This OSMLR descriptor is given an ID consisting of three parts:

1. The “level” of the segment. This is calculated from the road class of the first LRP. In level 0 are motorways, trunk and primary roads. In level 1 are secondary and tertiary roads. An optional level 2 can include all other roads, such as residential roads. (For the current purposes of the Open Traffic platform, we only generate segments for levels 0 and 1.)
2. The tile in which tile segment begins. Tile numbers are calculated as array indexes into a 2D array with the lower left at 0,0 = 0 and the lower right at x,0 = x, with the tile 1,0 = x+1. Tile sizes are 4 degrees on a side for level 0, 1 degree for level 1 and a quarter degree for level 2.
3. The ID within each tile. These are assigned to keep maximum compatibility between successive generations of data releases, such that each ID is immutable (only ever refers to a constant value of descriptor) although they may be marked as deleted if superseded in later data releases.

Relationship to OpenLR

These problems have already been faced and a solution found by the **OpenLR (http://openlr.net/)** method. Although OpenLR deals with many kinds of location references, Open Traffic only needs to specify linear references—i.e, sections of roads.

Therefore, the scheme described above is based on OpenLR with changes where appropriate to make it easier to work with OSM data.

Relationship to Valhalla

Valhalla (https://github.com/valhalla/valhalla) is an open source routing engine and accompanying libraries for use with OpenStreetMap data. Valhalla includes many routing-related tools, and it powers the hosted versions of **Mapzen Turn-by-Turn (/products/turn-by-turn/)** directions, **Mapzen Matrix (/blog/matrix)**, **Mapzen Optimized Route (/blogf/optimized-route)**, **Mapzen Isochrone (/blog/introducing-isochrone-service)**, and the **Mapzen Elevation (/blog/moving-on-up)** sampling service.

While the OSMLR model could be implemented independent from Valhalla, re-using existing, tested code to build the OSMLR segment generator reduces the size of the OSMLR codebase and makes it quicker and less error-prone to write. Since Valhalla is open source, there is little downside to introducing this dependency.

Creating OSMLR descriptors

The **OSMLR generator application (https://github.com/opentraffic/osmlr)** creates OSMLR descriptors from Valhalla road data tiles. It starts by merging across adjacent Valhalla graph edges which distinguish properties which are important to Valhalla's routing algorithms, but not to the collection of traffic data. For example, Valhalla keeps separate edges for bridges and tunnels as these can have important restrictions for traffic. However, since the start of a bridge or tunnel does not represent a point at which the driver of a vehicle has choice—if there is no junction—then for traffic purposes it can be counted as the same section of road.

For each merged sequence of edges, OSMLR will calculate the descriptor and output it in a tile. Calculating the descriptor involves splitting up the merged sequence into a set of LRPs such that the distance between neighboring LRPs is at most 15km. The properties are then calculated at each LRP, and stored in the descriptor.

Making sure that descriptors match only a single road is important, but currently unimplemented. This would mean matching the descriptor back to the road network and checking that it matched the original edges. Further, it is important to be *more* specific than necessary for the input dataset, as there may be differences between the input dataset from OSM and the same area at a later date, or a different source of data.

OSMLR descriptor packaging and distribution

OSMLR **descriptors (https://github.com/opentraffic/osmlr-tile-spec/blob/master/segment.proto)** are packaged into separate levels of **tiles (https://github.com/opentraffic/osmlr-tile-spec/blob/master/tile.proto)**, just like Valhalla road data tiles, which means that users can download only the subset of the data that they need. The descriptors and tiles are described in a **Google Protocol Buffers description (https://github.com/opentraffic/osmlr-tile-spec/)**. This makes the output files relatively compact, gives a clear path for upgrades to the format and makes it easier to construct tools and libraries to access that data from many different programming languages.

Keeping OSMLR IDs compact is important, as these IDs are used in many places to look up traffic data or to keep correspondences between datasets. To ensure that OSMLR IDs are compact while allowing for them to change over time, OSMLR IDs are assigned by a central process. When an updated set of OSMLR descriptors is wanted, the central process will match the previous set of descriptors against the new data and retain any which still match. This means that new IDs are only assigned to new roads, or ones which have undergone major changes. For major changes, the old IDs are marked as “deleted” so that the association between a road section and an OSMLR descriptor is kept unique for any given data release.

Matching OSMLR descriptors to a map dataset

OSMLR descriptors can be matched to a map dataset by taking the LRPs in each descriptor and making a shortest-distance route from each to the next coordinate, biasing in favor of routes which head in the right bearing. If several routes are candidates, then choose the one which most closely matches the descriptor.

This can be somewhat fuzzy, and there may be more than one match. In these cases, it may require some tie-breaking to figure out which is the best match. Descriptors may not overlap, so it’s possible to discard any part of the road network which was already matched to a descriptor. Indeed, descriptors which do not have a good candidate match—or have several—might be best postponed until all other good matches have been made to reduce the number of candidates. In general, a more major road such as a highway would make a better match than a residential road, but the circumstances may vary with the characteristics of the individual dataset.

Many of the descriptors will not be fuzzy, and will closely match a single segment of road. In these cases, optimizations are possible so as to not need to run a full route for each match. Running a route can sometimes be expensive in computer time, so avoiding it whenever possible is recommended.

The first case for optimization is when the target dataset has an edge in its routing network which corresponds exactly to the descriptor. These can be found easily when the source and destination points are the start and end nodes of an edge in the graph. While this may lead to spurious matches where there is also a shorter route, we rely on the extra attributes in the OSMLR descriptor—the bearing, length, road class, and so forth—to filter these out and ensure that only valid matches will be found.

This case can be extended for a sequence of edges. The edges leaving the start node can be “walked” to the next edges until either the end node is found, or a junction is encountered. Again, we rely on the properties of the descriptor to find candidate edges and filter out spurious matches.

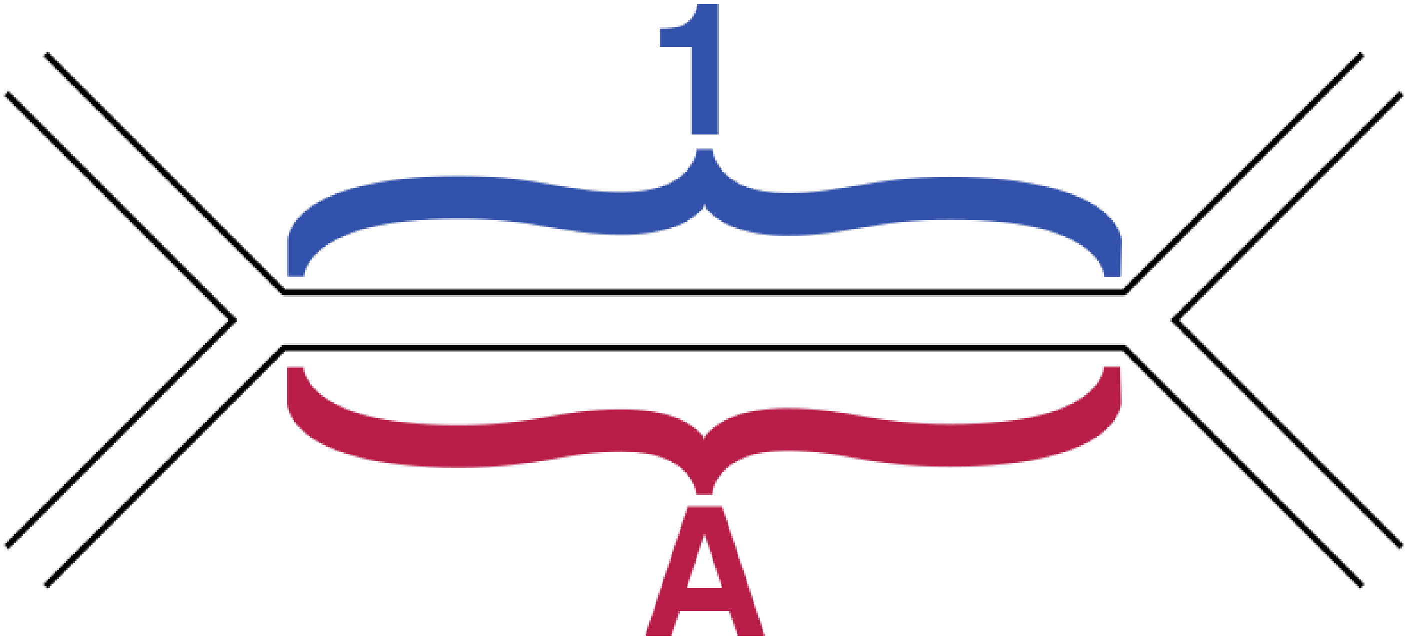
It is possible that several edges will match one descriptor, or that a single descriptor might only match a part of an edge. We call such collections of edges and descriptors “chunks.” Each chunk consists of multiple edges, each of which has several correspondences to ranges along some descriptors. A single edge might match the first 50% of its length to one descriptor and the remaining 50% to a different one, for example.

After the matching process is complete, this association between edges and OSMLR descriptors is saved to a data structure which handles both the common one-to-one match in a compact and efficient manner, but also allows for the more complex chunks. This consists of an array of OSMLR IDs indexed by the edge ID within a tile. There are some flag bits stored along with the OSMLR IDs which determine whether it is treated as a single ID for the common one-to-one case, or as an index into a second array of objects which describe the details of the chunk.

Different types of matches

One to One

The one-to-one match is the simplest case, and happily also the most common. In this case, there is one OSMLR segment describing a single graph edge in the map—whether that be an OSM way, Valhalla edge or data item from another map data source. This is illustrated below.

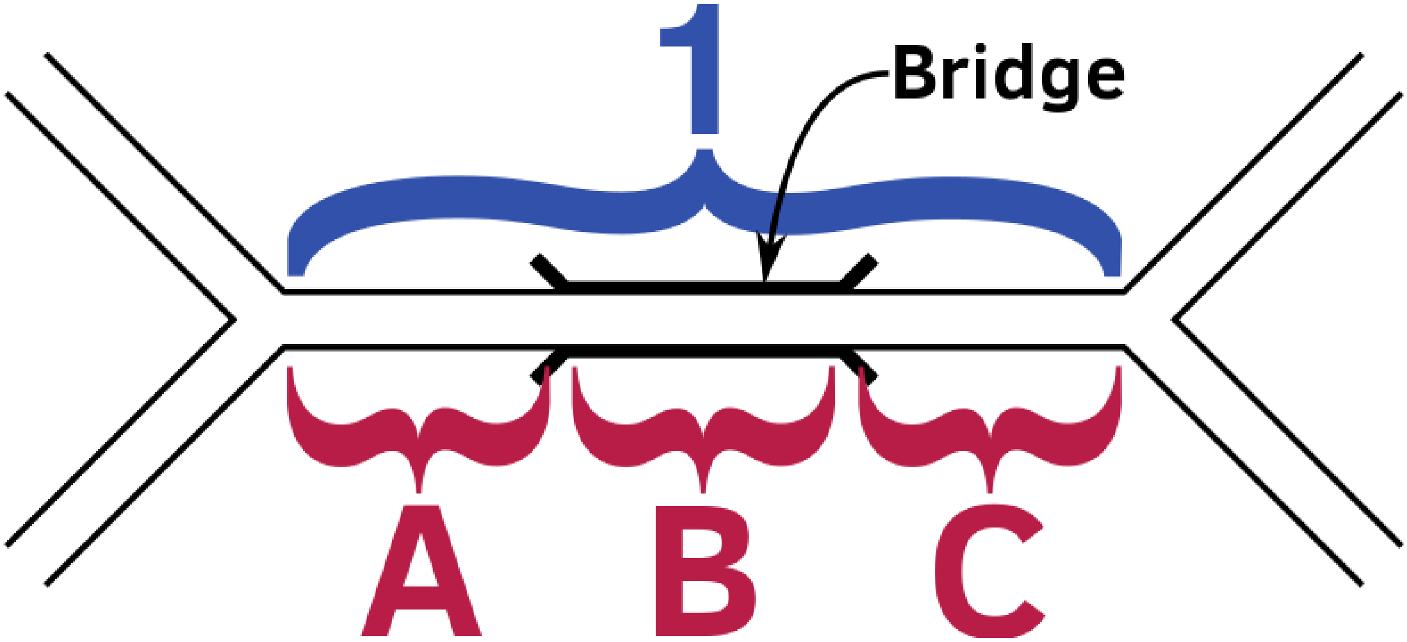


The OSMLR descriptor and the road in the data source overlap to within a tolerance. They start at similar points and end at similar points, and pass through similar points. The degree of tolerance that is used in the Valhalla matching process is 10 meters, although other values can be used depending on the particular circumstances of the dataset.

Given the one-to-one correspondence, it's easy to transfer properties—such as traffic speeds—from edges to OSMLR segments and vice-versa.

One to Many

In this case, one OSMLR segment describes several edges in the routing graph. This most commonly happens when an OSMLR segment elides a road feature which is unimportant for traffic purposes, such as a bridge.

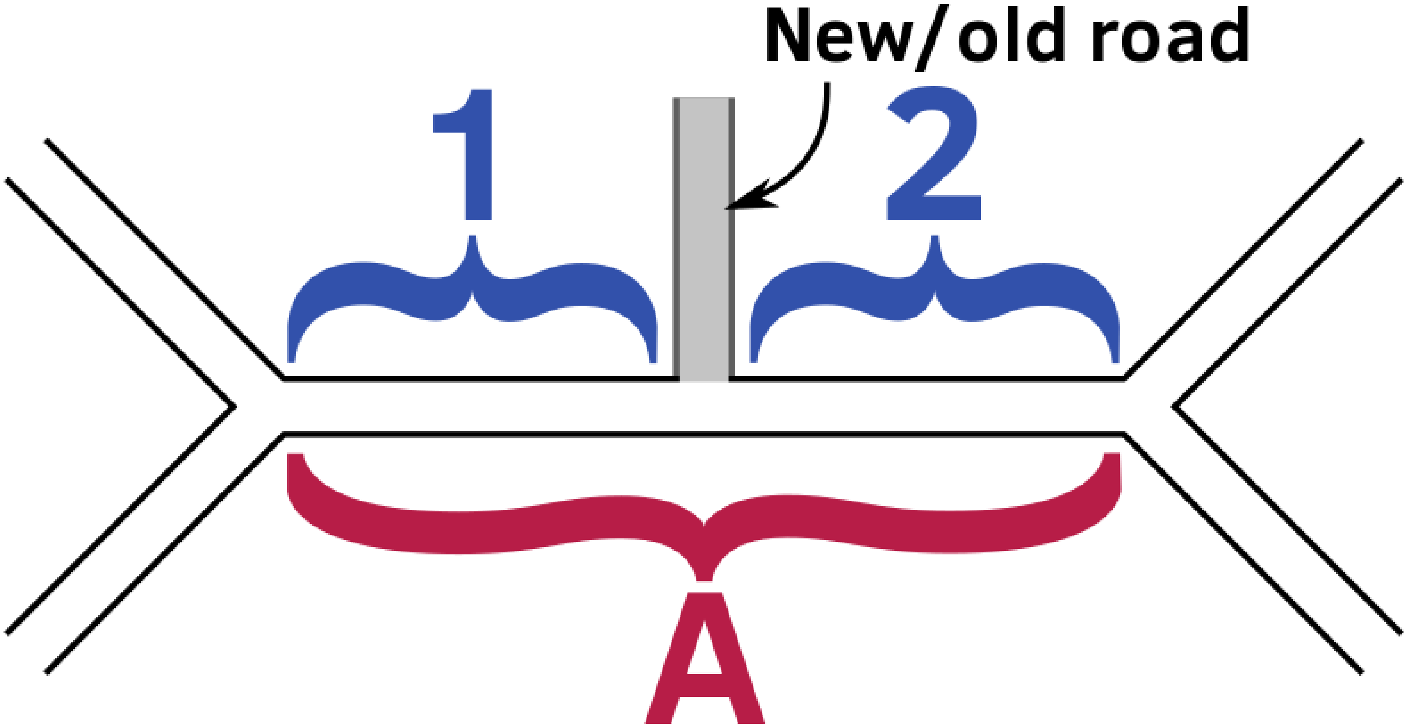


Although the edges partially overlap, the combination of the edges is equivalent to the OSMLR segment. The first node of the first edge matches the start of the OSMLR segment descriptor and the last node of the last segment matches the end of the OSMLR segment descriptor.

Transferring properties, such as traffic speed, from the descriptor to the edges might be as simple as assigning the same value to all of them. Transferring properties in reverse might require averaging, possibly with a length weighting.

Many to One

In this case, several OSMLR segments describe the same edges, although non-overlapping parts of it. This can happen when the data that the routing graph was generated from is different from the data used to generate the OSMLR segments. In the illustration below, this is represented by a road that has been added or deleted between the datasets such that it was present when the OSMLR descriptors were generated, and missing when the routing graph was generated.

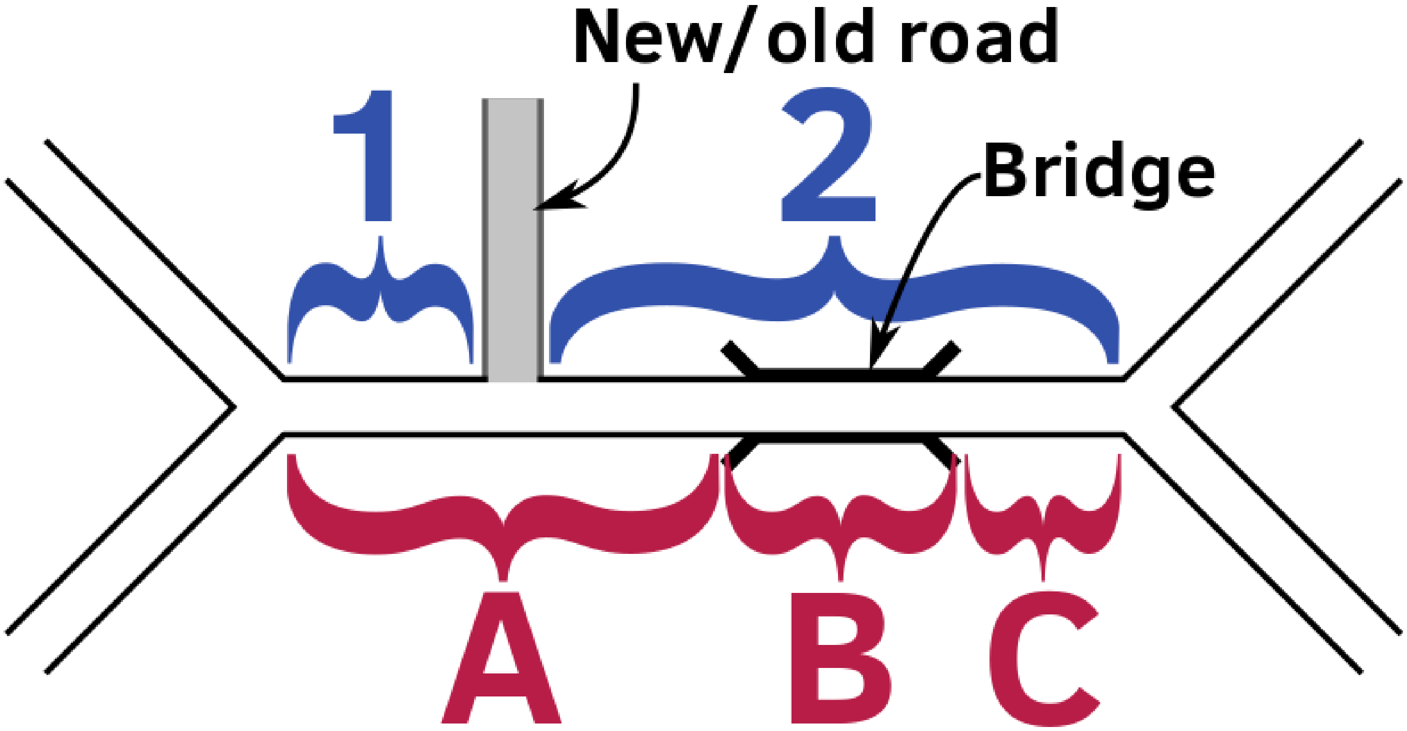


Each OSMLR descriptor matches only part of an edge, but the combination of all the descriptors matches the edge. The start of the first segment matches the first node, and the end of the last segment matches the end node.

Transferring properties is the inverse of the one-to-many case; going from the segments to the edge, one may want to do an average, weighted by length. When transferring from the edge to the segments, it may suffice to use the same value for all segments.

Many to Many

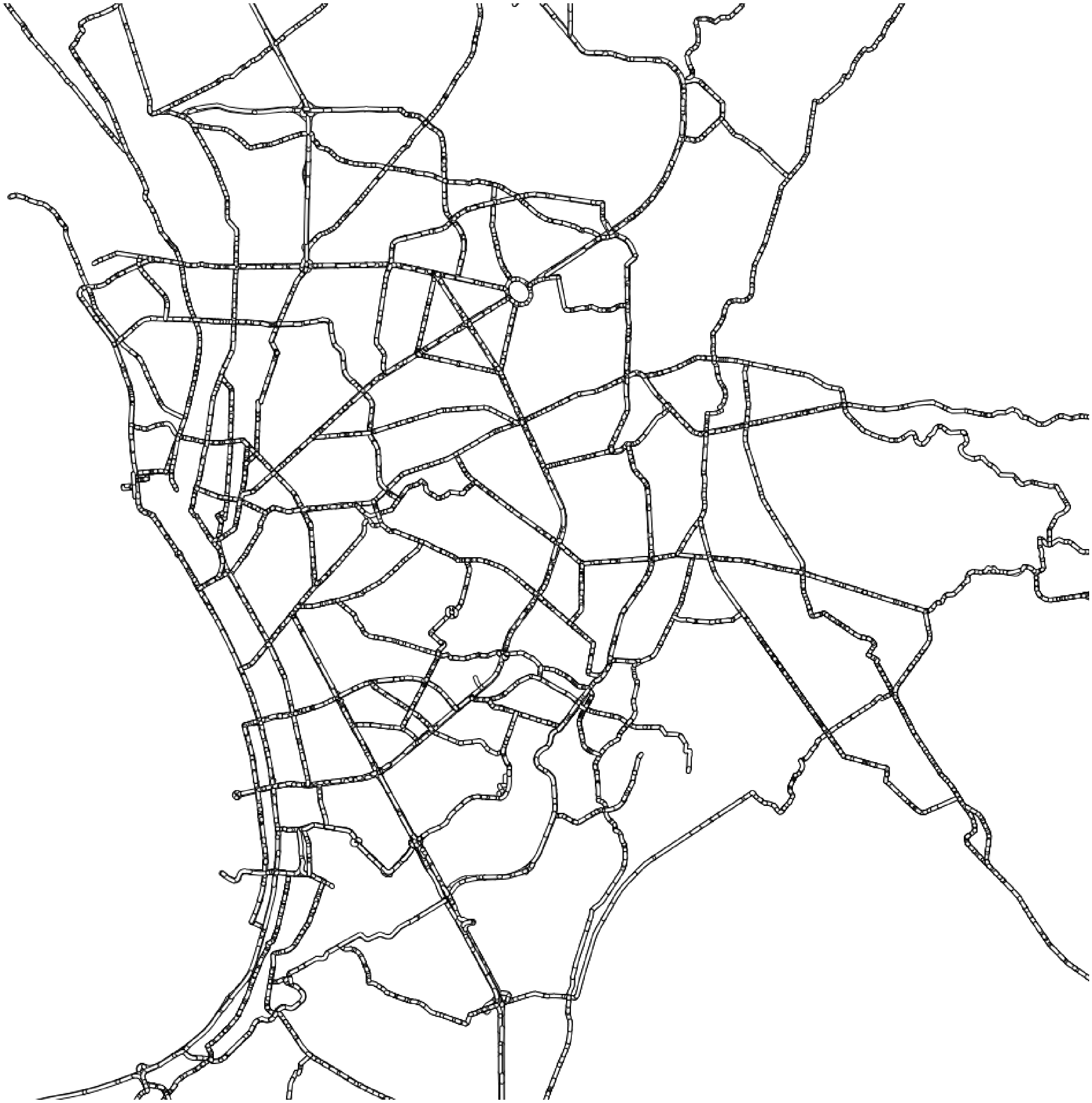
This case may arise out of a combination of the one-to-many and many-to-one cases, as illustrated below, or simply arise from complex differences between datasets. For the illustration, a road has been added or removed next to a bridge, and this means that the end points of the edge adjacent to the bridge and the new road do not coincide.



One way of dealing with these is to define “chunks”, with each chunk consisting of segments 1 and 2, plus edges A, B and C. While the interior points (between 1 & 2, or between A & B or B & C) are not shared between edges and segments, the start point of the chunk is shared between 1 & A, and the end point between 2 & C. This allows a linear reference along the whole length of the combined chunk, and so allows properties to be linearly interpolated, weighting by length.

How can I try OSMLR? And what’s next?

Eventually, the Open Traffic project will be serving out a single canonical source of OSMLR segments in tiled form for the entire world. In the meantime, we encourage you to look at the **opentraffic/osmlr-tile-spec repo on GitHub (https://github.com/opentraffic/osmlr-tile-spec)**, where you’ll find segment and tile schema definitions, along with samples of segment tiles that we’ve generated for Manila, Philippines. Samples are available in both protocol buffer format (for efficient processing) and as GeoJSON (for drawing geometries on a map):



Note that this version of OSMLR is a release candidate. We may need to make breaking changes in the future. We are also open to feedback and questions—please open an issue on GitHub on **opentraffic/osmlr-tile-spec (https://github.com/opentraffic/osmlr-tile-spec/issues)** (if it concerns the data schema) or **opentraffic/osmlr (https://github.com/opentraffic/osmlr/issues)** (if it concerns the generation logic).

Upon this foundation of stable segment identifiers, we're building the next stages of the Open Traffic platform to anonymize and aggregate traffic statistics. Please continue to watch the Mapzen blog for the next technical preview!

· 24 February 2017 ·

Matt Amos

OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

Kevin Kreiser

Kevin works on routing at Mapzen but secretly tries to work in all mapping disciplines. Er iss aa Pennsilfaanisch Deitscher.

David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

Mapzen Places API for Android

mobile (/tag/mobile) android (/tag/android)

Check out **Mapzen Android SDK version 1.3.0**. Not only does it contain bug fixes and performance improvements, but also introduces the **Mapzen Places API for Android** with place autocomplete, place picker, and place details!

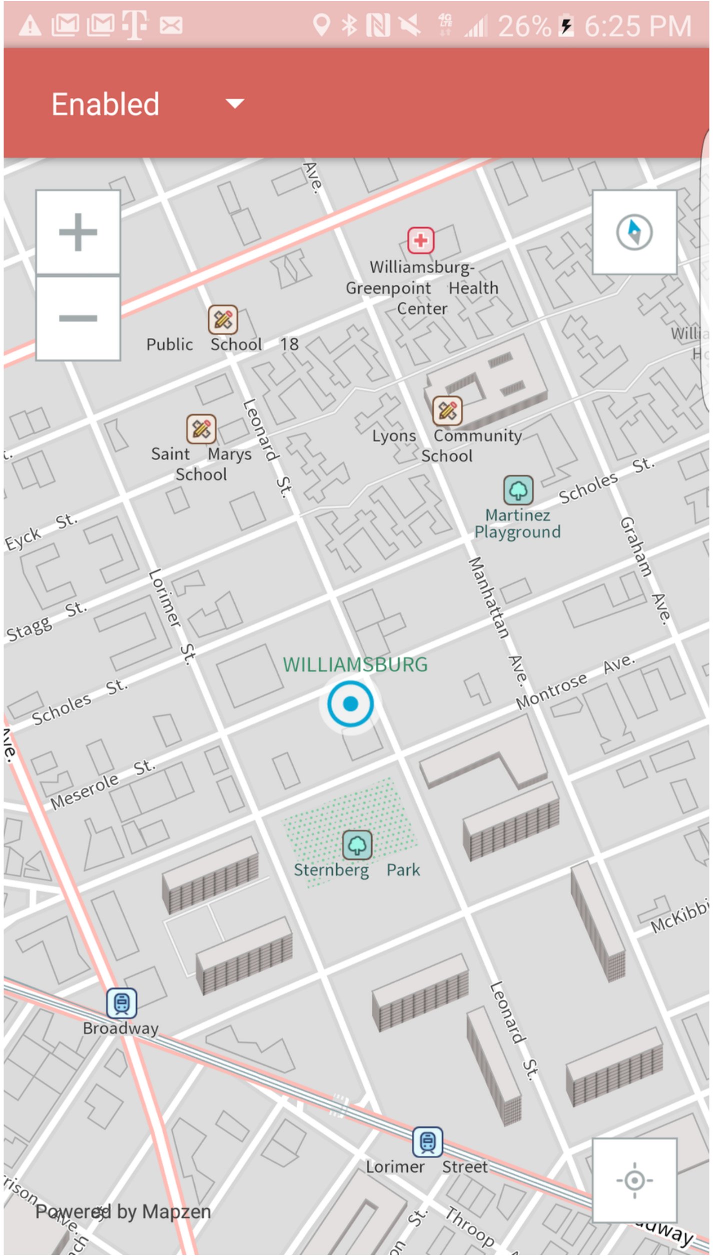
Mapzen Android SDK

Today we are announcing **version 1.3.0 of the Mapzen Android SDK**.

This release includes performance improvements and bugfixes for graphics, search, routing, and location services.

It also contains a number of new features including:

- Curved map labels
- New `SceneUpdate` interface for making changes to the map stylesheet at runtime
- Scene file parsing peformance improvements (up to 10x faster)
- Compass widget map overlay
- Zoom buttons map overlay
- Centralized Mapzen API key management


26% 6:25 PM
Enabled
Williamsburg-Greenpoint Health Center
Public School 18
Saint Marys School
Lyons Community School
Martinez Playground
Eyck St.
Stagg St.
Scholes St.
Meserole St.
Leonard St.
Lorimer St.
Manhattan Ave.
Graham Ave.
Montrose Ave.
WILLIAMSBURG
Sternberg Park
McKibbi
Broadway
Lorimer Street
Throop
Powered by Mapzen

For a full list of changes check out the **release notes (https://github.com/mapzen/android/releases/tag/v1.3.0)**.

As always the **Mapzen Android SDK** can be easily included in any Android app using Gradle.

```
dependencies {
  compile 'com.mapzen:mapzen-android-sdk:1.3.0'
  ...
}
```

Mapzen Places API

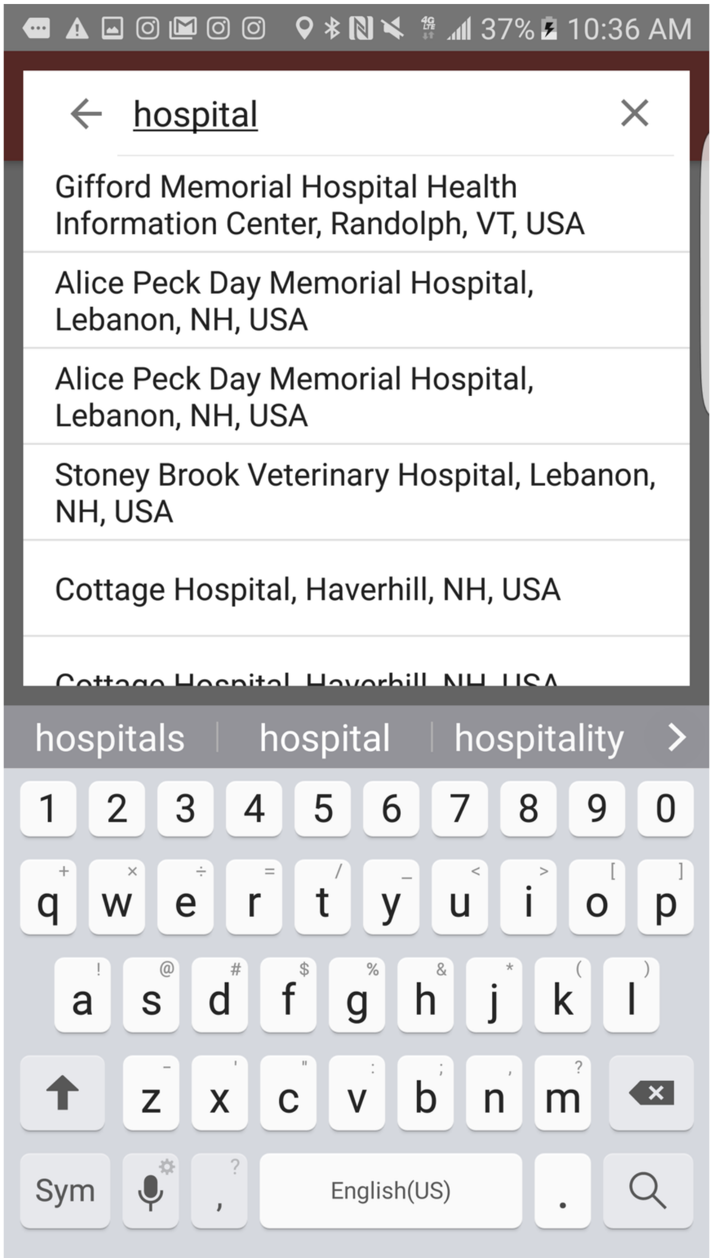
This release of the Mapzen Android SDK also introduces the new **Mapzen Places API for Android**.

The **Mapzen Places API for Android** combines Mapzen maps and geocoding into a seamless search experience to help you access open location information for places all around the world.

Place autocomplete

The **Place Autocomplete API** provides a search UI component that you can integrate into your app with a few lines of code. It is highly customizable and can be configured to only return search results with a precise address or results that are business establishments.

It can also be configured to be presented as an overlay or as a fullscreen component. When the user selects a result you will receive a callback with rich information about the selected place.


37% 10:36 AM
hospital
hospitals
hospital
hospitality
1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
Sym
English(US)

hospital

Gifford Memorial Hospital Health Information Center, Randolph, VT, USA

Alice Peck Day Memorial Hospital, Lebanon, NH, USA

Alice Peck Day Memorial Hospital, Lebanon, NH, USA

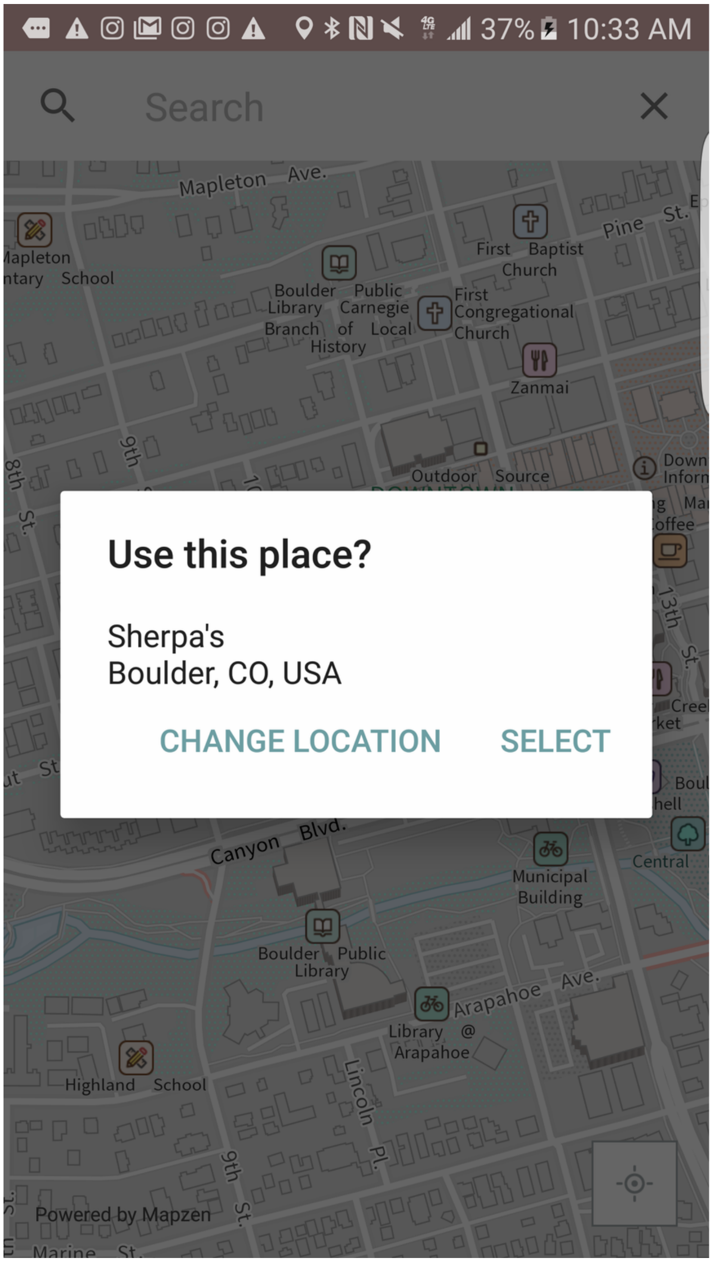
Stoney Brook Veterinary Hospital, Lebanon, NH, USA

Cottage Hospital, Haverhill, NH, USA

Place picker

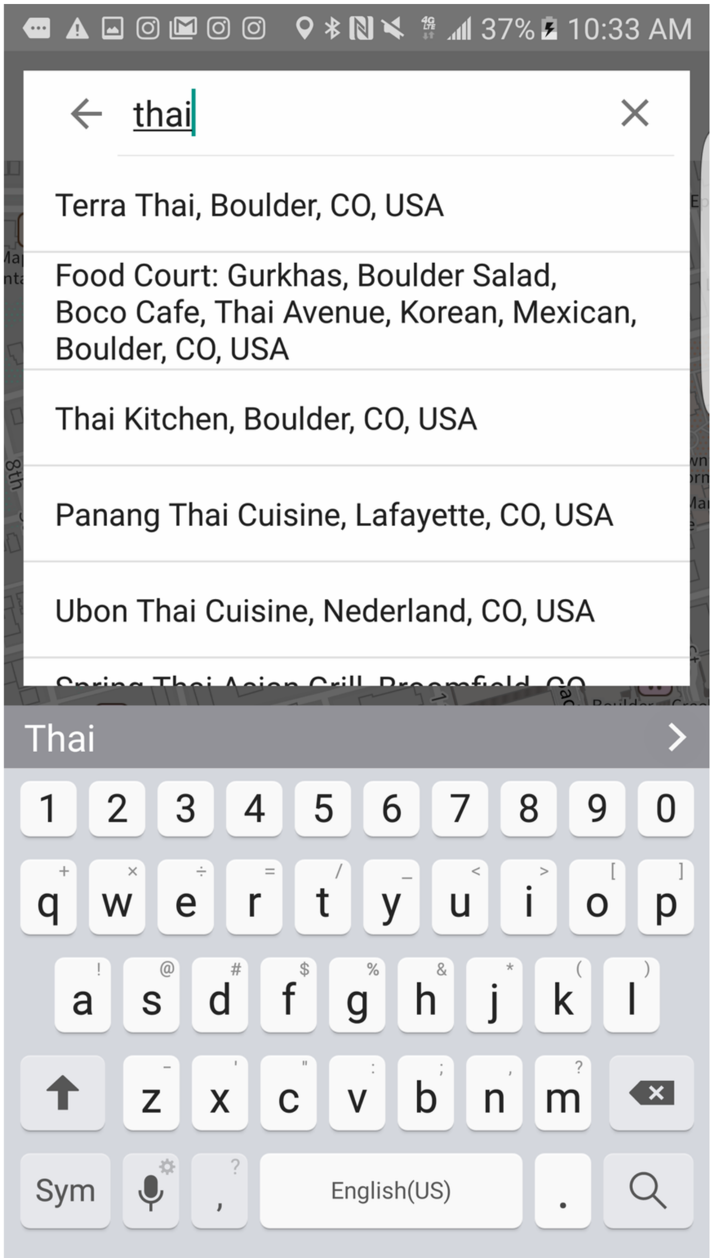
The **Place Picker API** combines the Place Autocomplete API with a map to allow place selection from multiple UI components.

Like the Place Autocomplete API, it is easily customizable and provides you with a full set of details about the selected place.


10:33 AM
Search
Mapleton Ave.
Boulder Public Library Carnegie Branch of Local History
First Baptist Church
First Congregational Church
Zanmai
Canyon Blvd.
Municipal Building
Boulder Public Library
Arapahoe Ave.
Highland School
Powered by Mapzen
CHANGE LOCATION
SELECT

Use this place?

Sherpa's
Boulder, CO, USA


10:33 AM
Thai
English(US)
Sym

thai

Terra Thai, Boulder, CO, USA

Food Court: Gurkhas, Boulder Salad, Boco Cafe, Thai Avenue, Korean, Mexican, Boulder, CO, USA

Thai Kitchen, Boulder, CO, USA

Panang Thai Cuisine, Lafayette, CO, USA

Ubon Thai Cuisine, Nederland, CO, USA

Place details

If you do not need a UI component but would like to receive place detail information, you can use the **GeoData API** to programmatically query for autocomplete results.

This API also allows for result filtering and returns place information with the same set of rich data used by the autocomplete widget and place picker.

For more information please refer to the **Mapzen Places API for Android documentation (https://mapzen.com/documentation/android/places/)**.

The **Mapzen Places API for Android** is available as a standalone library that can be included in your application by itself or alongside the **Mapzen Android SDK**.

```
dependencies {
  compile 'com.mapzen:mapzen-places-api:1.3.0'
  ...
}
```

Let us know (https://github.com/mapzen/android/issues) what you think!

· 03 March 2017 ·

Chuck Greb

Chuck is an open source enthusiast, test-driven evangelist, and clean code connoisseur of all things mobile.

Sarah Lensing

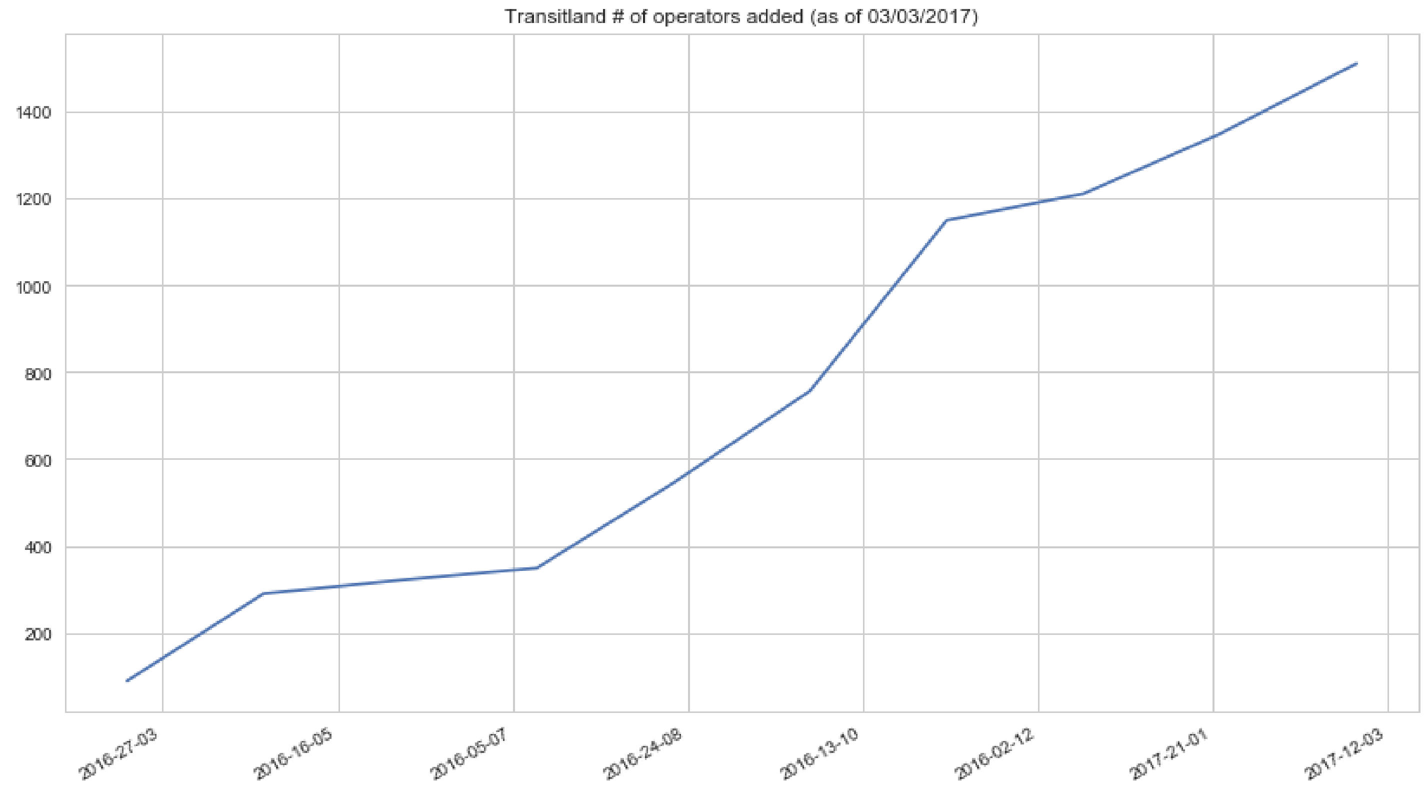
mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

© 2017 Mapzen

Transitland: One Year of Submissions

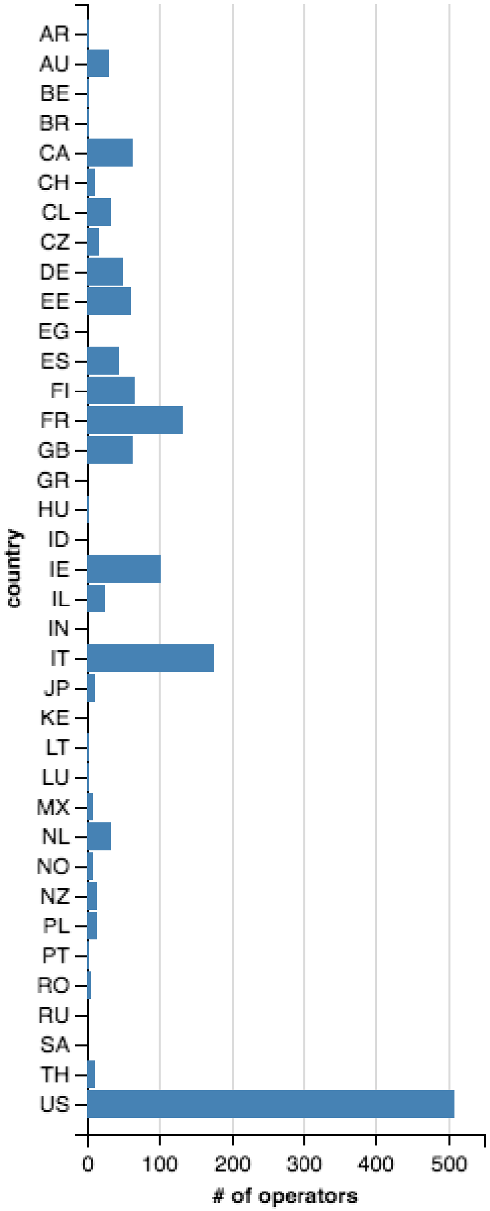
transitland (/tag/transitland) mobility (/tag/mobility)

We opened Transitland to submissions **a year ago (/blog/help-us-catalog-the-transit-feeds-of-the-world/)**, and boy have we come far together since then! As of today, Transitland includes data representing 1,511 transit operators:



(https://gist.github.com/drewda/0bcce89488612aeede93c424455dfdea)

These transit operators are located in 37 different countries:



(https://bl.ocks.org/drewda/3fdd43eff8db145a365f210a482f3914)

Actually, we should say that these operators are located in *at least* 37 different countries, since many systems extend beyond the borders of their headquarter's country.

As Transitland contributors stitch together coverage for new regions every day, the archipelago of interconnected transit islands grows into an ever larger network. This map shows tile squares where Mapzen Turn-by-Turn provides transit routing, using data provided by the Transitland Datastore. Each "island" of connectivity is displayed in a different color.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

*(**Open the world full screen (https://mapzen.com/tangram/view/?scene=https://mapzen.com/api/scenes/22/307/resources/basic.yaml#3/53.70/-37.62)**)*

Coverage and interconnectivity **is much more extensive than when we showed you this map last June (https://mapzen.com/blog/even-more-transit-routing/)**!

As you zoom in, the view will change to colored boxes outlining the service area of each operator. Names of operators will then appear. Zoom in further, and in certain cities like SF and NYC, the transit lines will appear along with their names and colors. There's a lot of data you can pull from the **Transitland API (https://transit.land/documentation/datastore/api-endpoints.html)** and **display in Tangram Play (https://mapzen.com/tangram/play/?scene=https://mapzen.com/api/scenes/22/307/resources/basic.yaml#3/53.70/-37.62)**!

These submissions are thanks to 188 users. A big "thank you" to all!

We'd also like to thank, in particular, **Steven Vance (http://www.stevevance.net/)** who has worked as Transitland's contract "data editor" since last summer. Steven reviews incoming submissions and helps to debug problematic feeds.

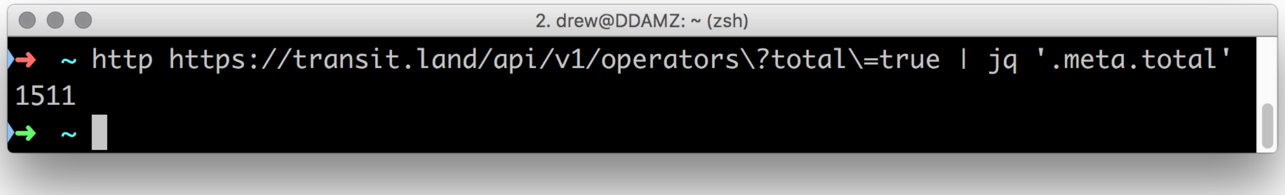
We're not finished. We welcome **your submissions of additional coverage to the Transitland Feed Registry (https://transit.land/feed-registry/feeds/new)**. It's a simple process, so simple that we have **a short video describing the steps (https://transit.land/documentation/feed-registry/add-a-feed.html)**.

If you're itching to start using all of this open transit data, try browsing using the the **Mapzen Mobility Explorer (/blog/introducing-mobility-explorer/)**, plan a transit route using **Mapzen Turn-by-Turn (/products/turn-by-turn)**, or read about the **Transitland Datastore API (https://transit.land/documentation)** that powers those and other applications. You can also try adding the GeoJSON Transitland API response as a data source **in Tangram Play (https://mapzen.com/tangram/play/?scene=https://mapzen.com/api/scenes/22/307/resources/basic.yaml#3/53.70/-37.62)**.

In fact, both of these charts were made using calls to the Transitland Datastore API, and like most everything at Mapzen, they're based on open-source code:

- **Jupyter Notebook with Python code to generate the operators-by-created-date plot (https://gist.github.com/drewda/0bcce89488612aeede93c424455dfdea)**
- **JavaScript code to generate the operators-by-country plot using d3/Vega Lite (https://bl.ocks.org/drewda/3fdd43eff8db145a365f210a482f3914)**

But if you're impatient, don't want to plot out a whole graph, and just want to know the current number of operators available in Transitland as it keeps on going up, here's an easy way to get that from your terminal. Using the **httpie (https://httpie.org/)** and **jq (https://stedolan.github.io/jq/)** libraries:



```
➜  ~ http https://transit.land/api/v1/operators\?total\=true | jq '.meta.total'
1511
➜  ~ 
```

And finally, as always, you can browse through all the operators in Transitland using the **Transitland Feed Registry (https://transit.land/feed-registry)**.

We've enjoyed this past year of working together on aggregating open transit data from around the world—and we look forward to continuing this impressive growth together!

· 06 March 2017 ·

Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

Meghan Hade

Meghan is a software engineer with a background in urban planning on Mapzen's Mobility team. She works in javascript and plays in calligraphy, block printing, and finding ways to stash n+1 bikes in a San Francisco apartment.

Ian Rees

Ian spends too much time thinking about cities, land use, and open transit data on the Mapzen Mobility team.

Alex Smith

© 2017 Mapzen

OpenAddresses celebrates 400 million addresses

data (/tag/data) search (/tag/search) geocoding (/tag/geocoding)

In our recent **interpolation (https://mapzen.com/blog/interpolation/)** blog post, Peter mentioned how address geocoding in **Mapzen Search (https://mapzen.com/products/search/)** is powered by two important open-source projects: **OpenStreetMap (http://www.openstreetmap.com/)** and **OpenAddresses (http://www.openaddresses.io/)**.

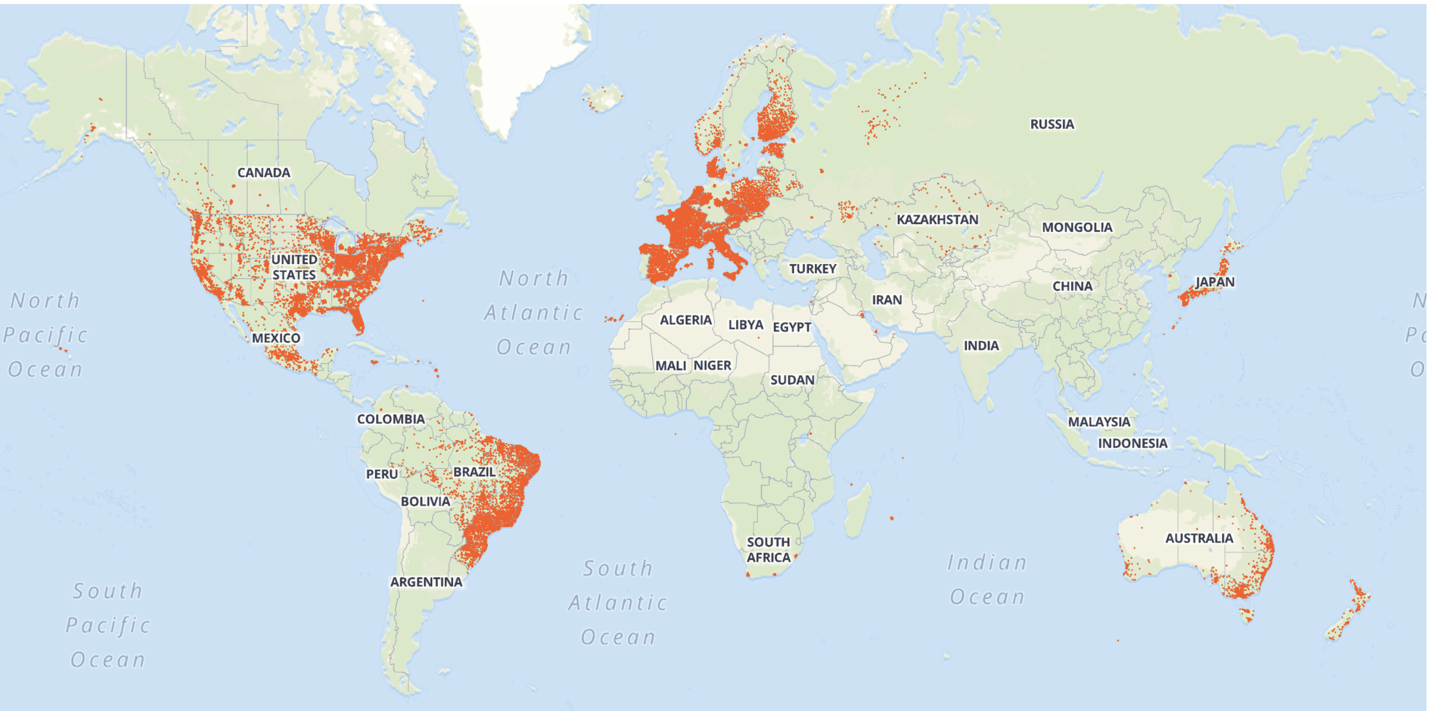
Mapzen is proud to help improve **OpenAddresses** data coverage and we'd like to celebrate two important milestones for the project. OpenAddresses surpassed 400 million address in January 2017, and we are on track to add the 2,000th source in March. That's 8.2GB of open-data! 🎉

A short history of OpenAddresses

OpenAddresses **launched (https://web.archive.org/web/20160305125450/http://openaddresses.tumblr.com/post/80586010795/the-future-of-addresses-is-open-launching)** in March 2014. Street addresses are sourced directly from the government agencies responsible for creating and maintaining cadastral information.

While OpenStreetMap has address coverage in **more places (https://taginfo.openstreetmap.org/keys/addr%3Ahousenumber#map)**, OpenAddresses covers six times the number of addresses overall and often has *more* complete coverage in those *fewer* places. (If you need to add a missing address, OpenStreetMap is the best place to contribute.)

Together these complimentary projects have nearly half a billion addresses!


CANADA
UNITED STATES
MEXICO
North Pacific Ocean
North Atlantic Ocean
COLOMBIA
PERU
BRAZIL
BOLIVIA
ARGENTINA
South Pacific Ocean
South Atlantic Ocean
ALGERIA
LIBYA
EGYPT
MALI
NIGER
SUDAN
SOUTH AFRICA
TURKEY
IRAN
KAZAKHSTAN
RUSSIA
MONGOLIA
CHINA
JAPAN
INDIA
MALAYSIA
INDONESIA
AUSTRALIA
Indian Ocean

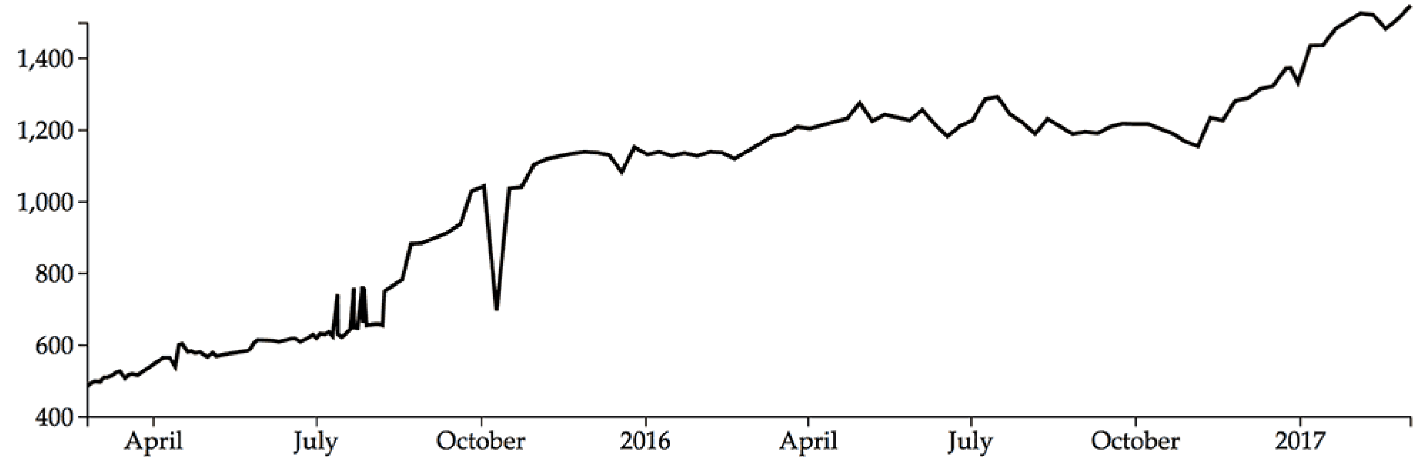
March 2017: *406 million addresses (up from* ***200 million in July 2015 (https://mapzen-assets.s3.amazonaws.com/images/2017-03-07-open-addresses-400mm/openaddresses-2015-07-dotmap.gif)****!)*

Ian Dees co-founded OpenAddresses with **Nick Ingalls (https://github.com/ingalls)**, and **Mike Migurski** has been an important contributor. Major props also to our friends at **Mapbox (http://mapbox.com)** for help with the OpenAddresses project, and to Esri who enable easy access to address and parcel features in over **4,000 (https://twitter.com/ajturner/status/838446044891365377)** government open-data portals!

After a period of rapid expansion in 2015, the addition of new OpenAddresses sources started to taper off. Most of the “easy” sources were played out and keeping existing sources fresh became its own task. The project hit 200 million addresses with 1,000 sources in October of 2015.

There were still some big new sources added to OpenAddresses in 2016, including national datasets in Australia, Italy, Mexico (where Mapbox is currently working on another huge update), Austria, parts of Germany, parts of Switzerland, Finland, Kazakstan, parts of Russia, and several major cities around the world. The United States saw some improvement in early and mid-2016, and the pace has picked up quite a bit the last 6 months.

Globally, the 300 million mark was reached in December 2016, and in January 2017 a contributor added nationwide coverage for Brazil, bumping worldwide coverage up to 400 million addresses.



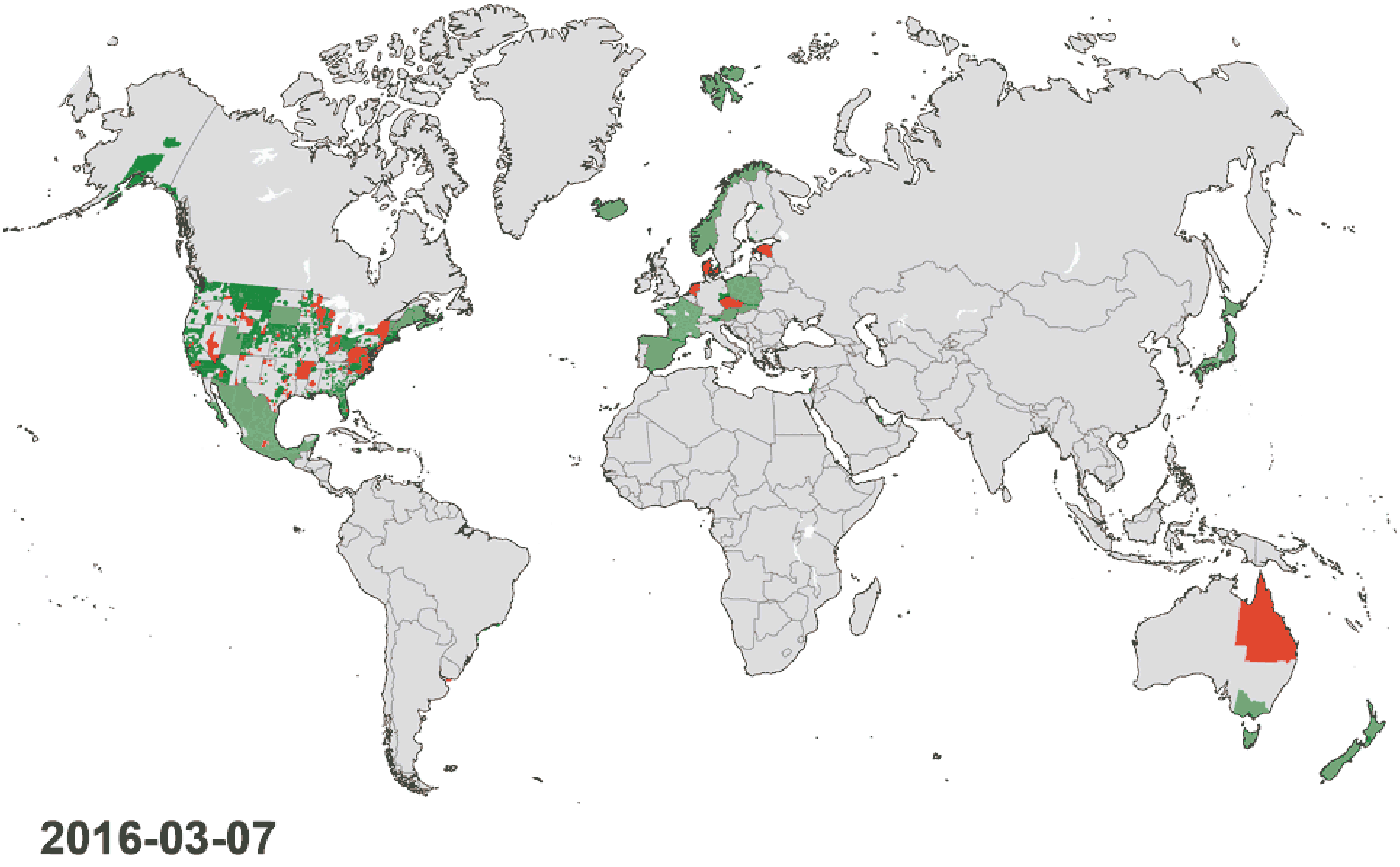
*The number of **successful sources (http://results.openaddresses.io/dashboard)** per build is surging. There are currently **1965 total sources (https://results.openaddresses.io/sets/162276/)**.*

Mapzen + OpenAddresses

Stephen Hess with Mapzen Search started helping out with OpenAddresses in October of 2015 for Pelias, landing his **first Pull Request (https://github.com/openaddresses/openaddresses/pull/1393)** in November 2015. Since then he’s added many new sources, gardening existing sources to standardize their configurations and refreshing them when they go stale. Improving documentation (including a section on using `regex` to magic data out of difficult sources) has also been a focus. **Ian** and **Mike** have helped with pull requests for sources and kept the OA machine humming.

In the fall of 2016 **Nathaniel Vaughn Kelso** inventoried coverage in the United States and started filing issues documenting remaining major gaps.

That work is organized in several Github projects **1 (https://github.com/openaddresses/openaddresses/projects/1)**, **2 (https://github.com/openaddresses/openaddresses/projects/2)**, and **3 (https://github.com/openaddresses/openaddresses/projects/3)** around four priority tiers of (1) major cities and counties, (2) medium sized cities and counties, (3) regionally important cities and counties, and (4) small towns and rural counties. After that initial round of Github issues was filed, we received many tips via issue comments pointing to possible data sources —thank you!



Animated map: Progress adding sources during the last 12 months globally and in the United States of America. Learn about the orange sections below.

Starting in September, 2016 we have been lucky to work with **Justin Meyers (https://github.com/justinelliotmeyers)** as a data prospector to improve coverage in the United States. The pace picked up quite a bit in November once we got fully online, and we've found a few international sources along the way.

Workflow

Most missing sources take just 15-30 minutes to research on the web and add to OpenAddresses with a pull request via Github, where the project is managed.

Single data sources are often at the locality or county level with **10,000 to 200,000 (http://results.openaddresses.io/dashboard)** addresses. But sometimes we get lucky and can add millions of addresses with an entire state (like in **Texas (https://github.com/openaddresses/openaddresses/pull/2180)**), or a metro area (like **Detroit (https://github.com/openaddresses/openaddresses/pull/2295)**) spanning multiple counties.

For a lot of the smaller counties in the United States we pick up the phone and dial the local planning department or tax accessor to find the data. In some cases we've found the sheriff's office has addresses as part of a federally sponsored **e911 (https://en.wikipedia.org/wiki/Enhanced_9-1-1)** project.

There are a few open-data holdouts (we're looking at you, Philly suburbs) where the data is available, but under closed terms and requires a huge fee. Hopefully the Pennsylvania governor's new **open-data initiative (https://www.governor.pa.gov/governor-wolf-announces-open-data-initiative-to-engage-citizens-in-innovative-policy-solutions/)** will help.

Most data is available as a web service or download, but sometimes the data provider will ship us a DVD!

County of Sierra
Laura A. Marshall, Assessor
P.O. Box 8
Downieville, CA 95936
USPS

Purchasing data for Sierra County, California via snail mail. We had to buy a DVD reader. 📀

Contributing

In the last 12 months a lot of data has been added, jumping roughly from ~240 million in February 2016, to ~275 million in September 2016, and to 406 million addresses today. The best part is over 100 million addresses have come from people besides us, including: **davidchiles (https://github.com/davidchiles)**, **astoff (https://github.com/astoff)**, **sergiyprotsiv (https://github.com/sergiyprotsiv)**, and **thatdatabaseguy (https://github.com/thatdatabaseguy)**. Thank you! 😃

OpenAddresses would love your help. The **contributing guidelines (https://github.com/openaddresses/openaddresses/blob/master/CONTRIBUTING.md)** is a good place to start.

Curious about data gaps? Use OpenAddresses's **dot map (https://openaddresses.io/#map)** to browse data coverage. If you see a gap look thru the **issues (https://github.com/openaddresses/openaddresses/issues)** to see if someone is already working on it. When you find a data source for the missing place, add your research to an **existing issue (https://github.com/openaddresses/openaddresses/issues)** or create a **new issue (https://github.com/openaddresses/openaddresses/issues/new)**.

Are you more technically inclined? Create a JSON config file for new sources for issues marked **ready for PR (https://github.com/openaddresses/openaddresses/issues?q=is%3Aissue+is%3Aopen+label%3A%22ready+for+PR%22)**, or look through sources which have **gone stale (http://results.openaddresses.io/dashboard)** (listed as "Lost sources" and colored orangeish-red in the animated maps above) and give them some love by adding a missing **conform (https://github.com/openaddresses/openaddresses/blob/master/CONTRIBUTING.md#conform-object)**.

And if you are available to user test some new features we are building please say **hello@mapzen.com (mailto:hello@mapzen.com)**.

Happy mapping!

· 07 March 2017 ·

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

Stephen Hess

Stephen works on geocoding exclusively as a means to fund his passion for designing and building wooden frames for old maps.

Justin Meyers

Justin is a GIS data guru, specializing in administrative and international boundary data. Otherwise, his life revolves around his four german shepherds.

Ian Dees

Ian is on the tiles team, building squares of map data. Ian ❤️maps and bleeds nodes, ways, and relations.

Michal Migurski

Oakland-dwelling geodata cyclist.

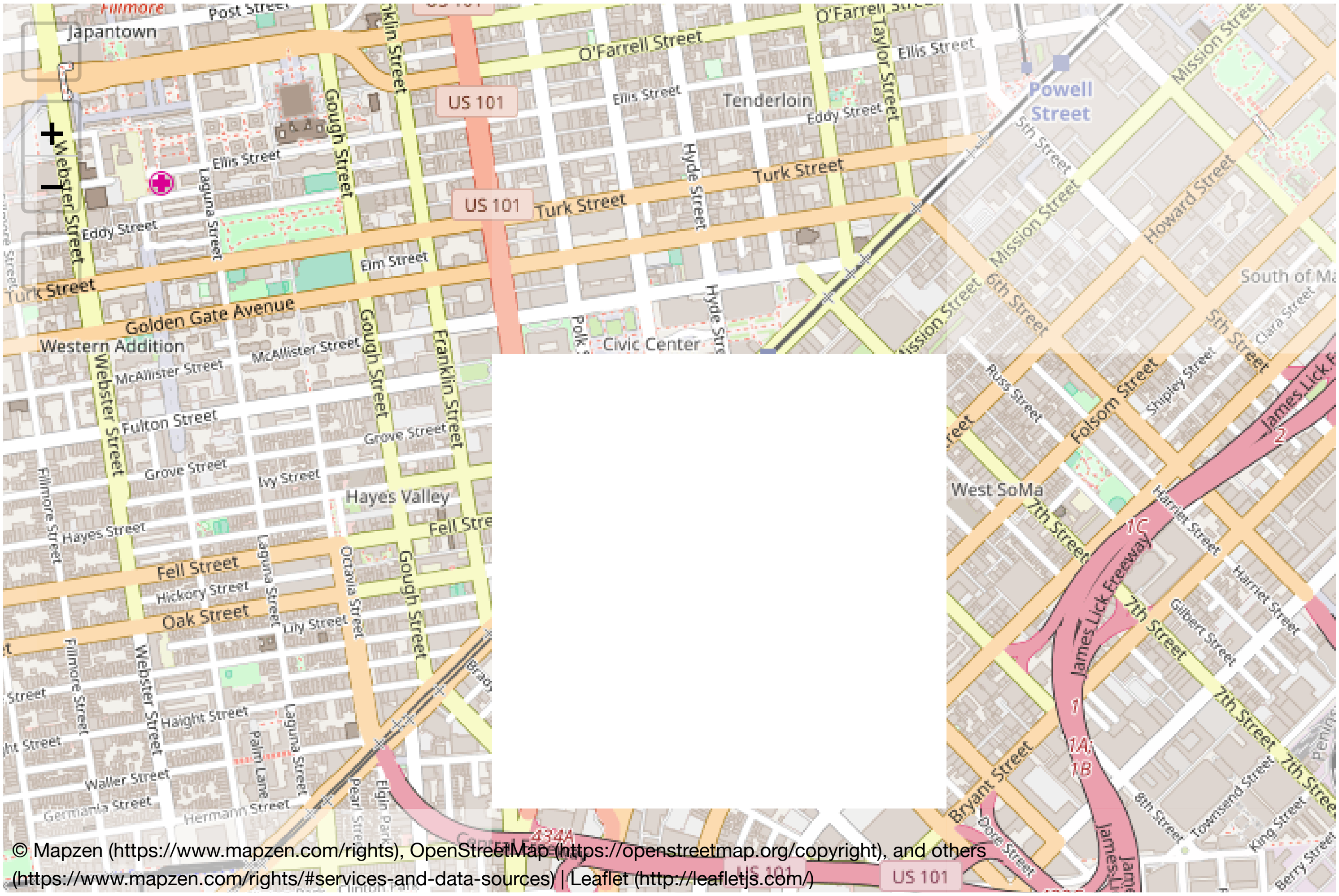
© 2017 Mapzen

Spring into cycling with Mapzen's new bike map

cartography (/tag/cartography) tangram (/tag/tangram) vector-tiles (/tag/vector-tiles)

We heard your comments and updated the bike map, thank you! **Checkout the new version... (http://mapzen.com/blog/bike-map-v2)**

Sometimes finding a bike route can be as hard as biking up a hill. Our new **bike overlay** for Walkabout shows the way, and is enabled by fantastic bike data in **OpenStreetMap (https://www.openstreetmap.org)** and **Tangram (https://mapzen.com/documentation/tangram/)**’s shader magic. If you’ve ever edited `cycleway` tags in OpenStreetMap, this map is dedicated to you. 🚲


Japantown
Post Street
O'Farrell Street
Tenderloin
Powell Street
Ellis Street
Eddy Street
Turk Street
US 101
Golden Gate Avenue
Western Addition
McAllister Street
Civic Center
Fulton Street
Grove Street
Hayes Valley
Hayes Street
Fell Street
Oak Street
Haight Street
Waller Street
Hermann Street
Webster Street
Laguna Street
Gough Street
Franklin Street
Hyde Street
Taylor Street
Mission Street
Howard Street
Folsom Street
South of Market
West SoMa
6th Street
7th Street
8th Street
Harriet Street
Bryant Street
James Lick Freeway
Townsend Street
King Street

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

(This map is interactive! ***Open full screen ➶ (https://mapzen.com/products/maps/walkabout/bike-map/)*** *)*

As usual, you can enable the bike overlay **in just a few lines**!

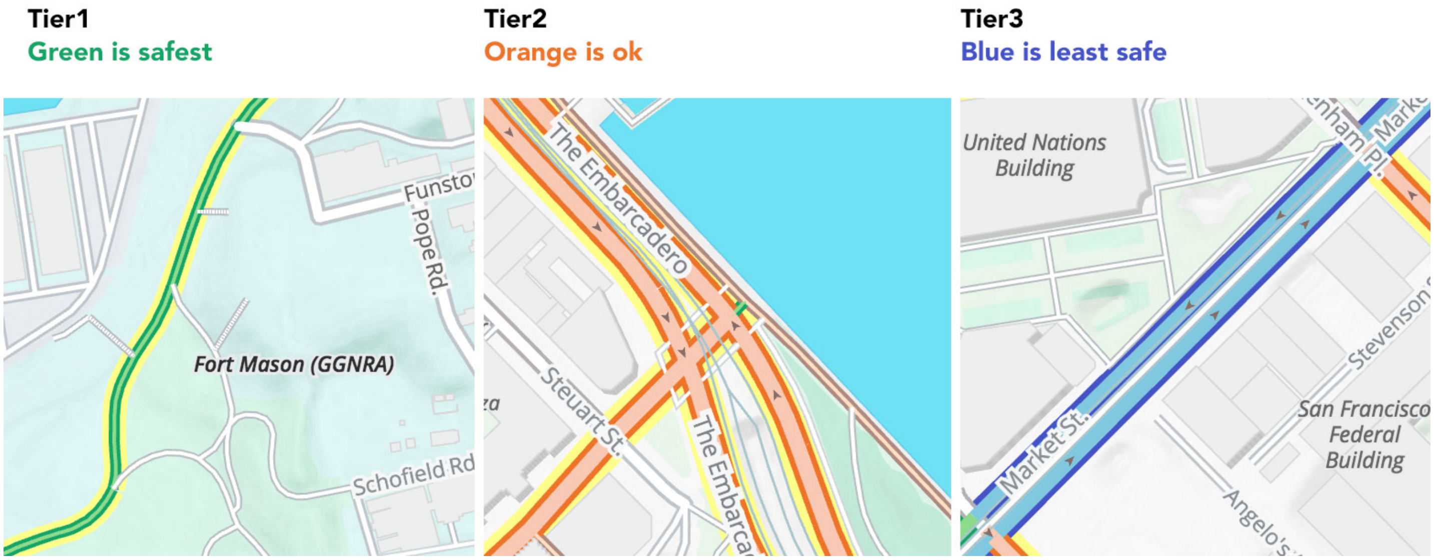
In preparation for the new bike overlay, we've also made some other improvements to Walkabout, particularly on roads, which we'll go over in more detail below.

But first, let's review how we're presenting the data in the new bike overlay. Cycle data gets **very complex very quickly (https://wiki.openstreetmap.org/wiki/Bicycle)**, and after lots of sketching, we decided it would be best to organize the data according to safety. What is the safest route? Do I have to ride with cars? Which side of the road can I ride on?

These are some of the questions we asked when we created these categories and in the design process of the new bike overlay. Let's review these categories below.

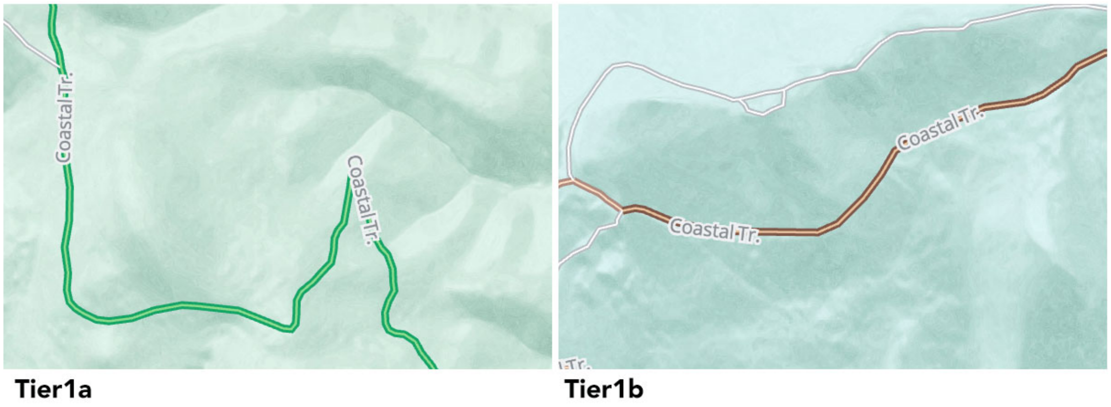
Where to Bike

Roads and paths have been categorized into three tiers based on **safety**.

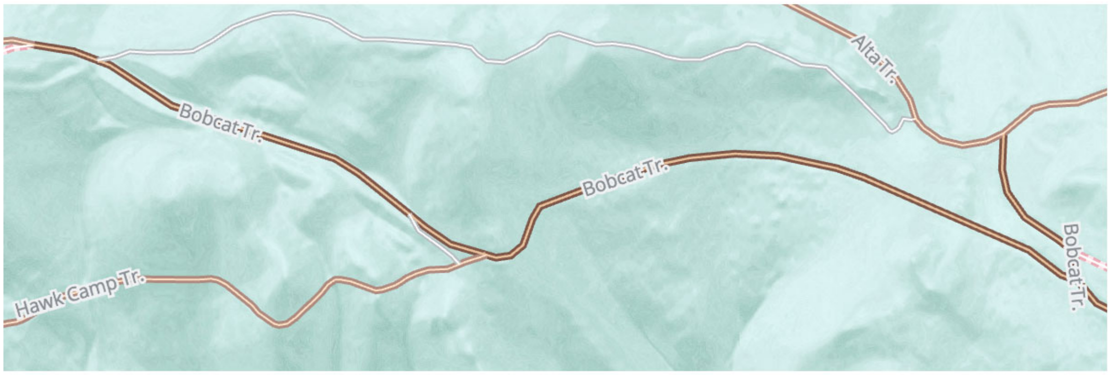

Tier1
Green is safest
Tier2
Orange is ok
Tier3
Blue is least safe
Funston
Pope Rd.
Fort Mason (GGNRA)
Schofield Rd
The Embarcadero
Steuart St.
The Embarca
United Nations Building
Market St.
Stevenson
San Francisco Federal Building
Angelo's

Tier 1 is further divided into 2:

- **Tier 1a** off-road paths in **GREEN** *(no cars)*
- **Tier 1b** off-road dirt track in **BROWN** *(no cars)*

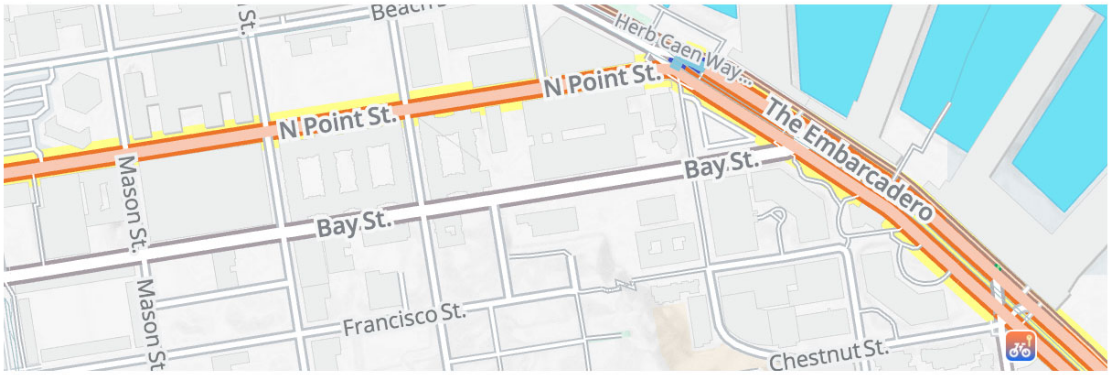

Coastal Tr.
Coastal Tr.
Tier1a
Coastal Tr.
Coastal Tr.
Tier1b

Tracks also include footways which are displayed in a lighter shade of brown, like Hawk Camp Trail and Alta Trail as footways compared to Bobcat Trail as a track.


Alta Tr.
Bobcat Tr.
Bobcat Tr.
Bobcat Tr.
Hawk Camp Tr.

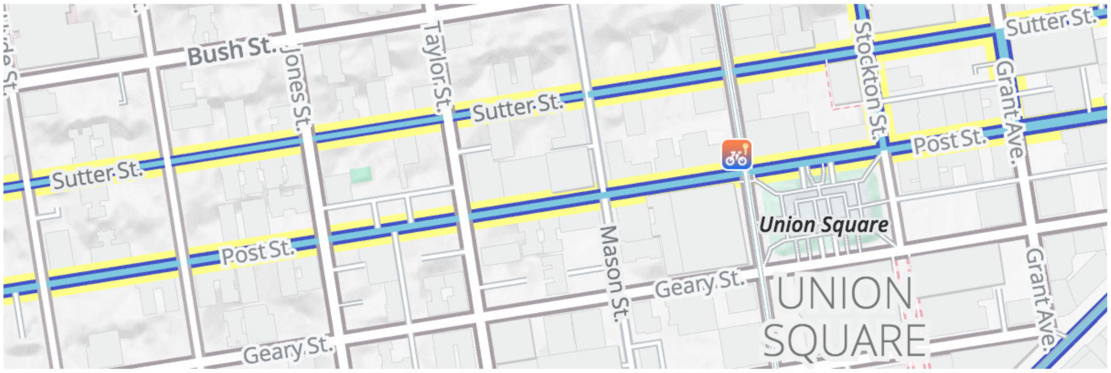
Tier 2 - **ORANGE**

- on-road painted lanes *(ride your bike next to cars)*

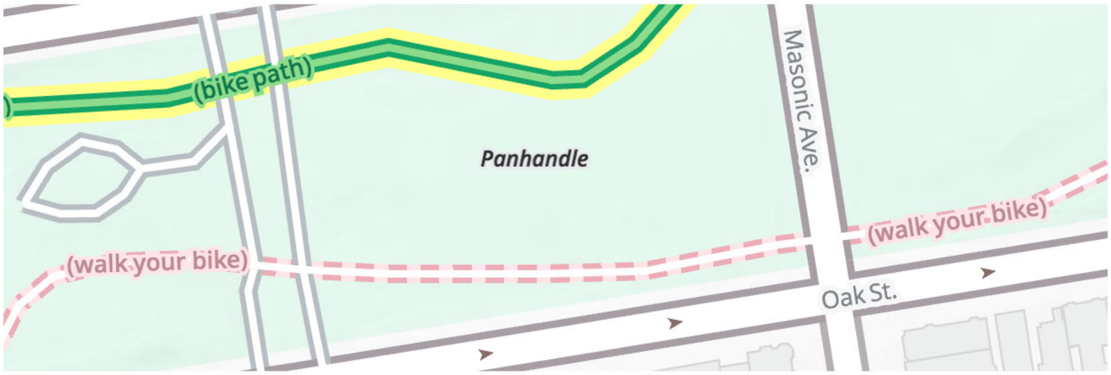

St.
Beach
Herb Caen Way...
N Point St.
N Point St.
The Embarcadero
Mason St.
Bay St.
Bay St.
Mason St
Francisco St.
Chestnut St.

Tier 3 - **BLUE**

- on-road “route” signage *(ride your bike with cars)*


Bush St.
Jones St.
Taylor St.
Sutter St.
Stockton St.
Sutter St.
Grant Ave.
Sutter St.
Post St.
Union Square
Post St.
Mason St.
Geary St.
UNION SQUARE
Grant Ave.
Geary St.

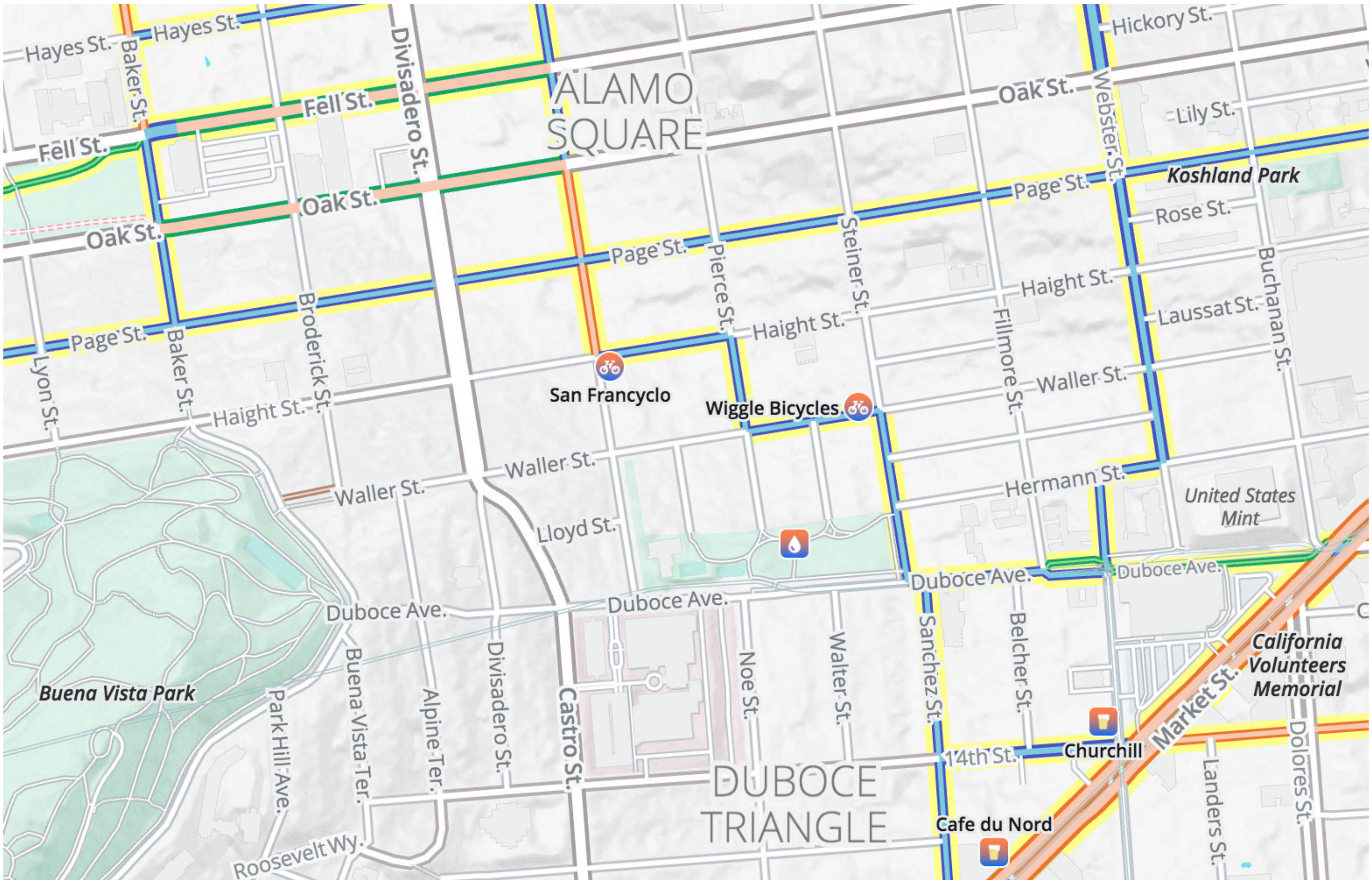
And there is a fourth tier, **Tier 4** for bicycle=no or “walk your bike” in pink dashed lines.



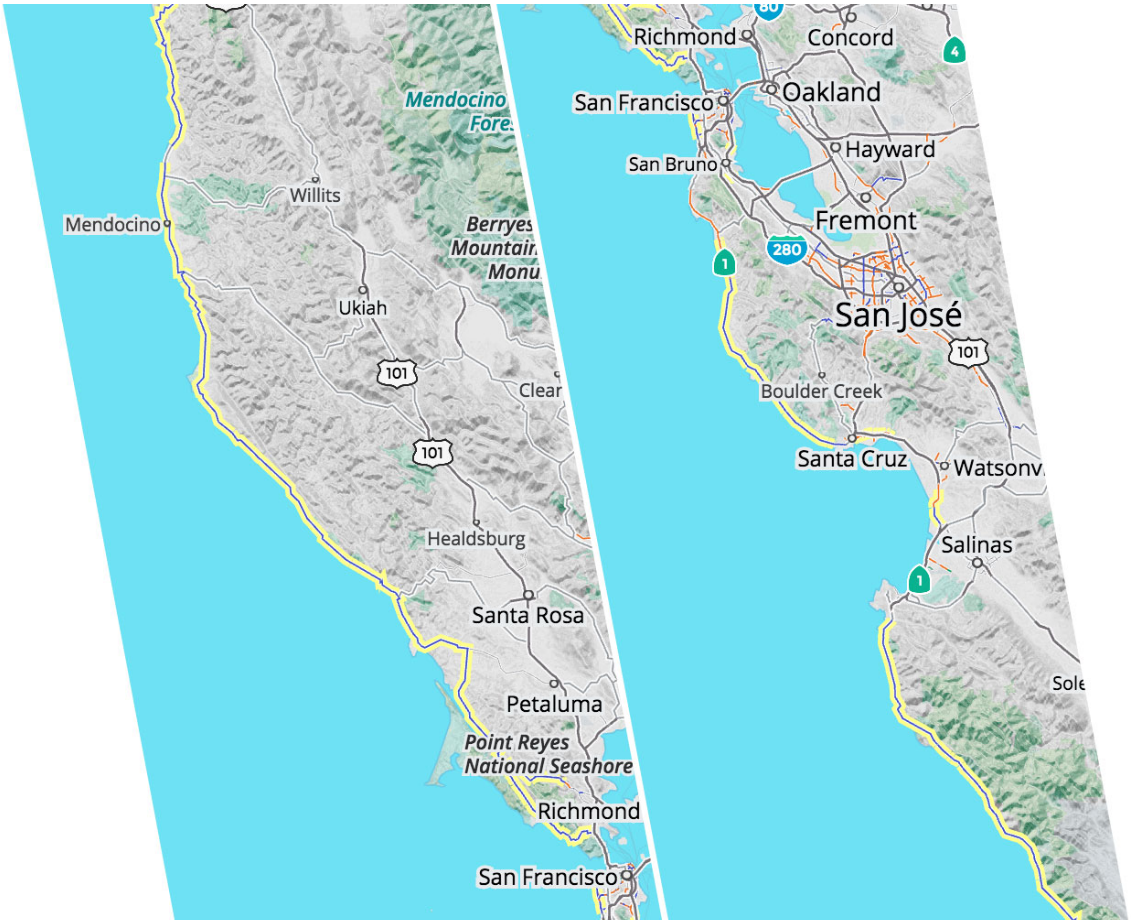
Roads without any bicycle facilities are drawn in white with grey casing. If a road has bike facilities in real life (or bikes are prohibited) but that information is missing on the map, learn how to tag the roads in OpenStreetMap in the **Powered By Mapzen Vector Tiles** section below.

Bike Networks

A **yellow highlight** is added to roads when they are part of a bike network. For example, there's a popular San Francisco bike network called **the Wiggle (http://www.sfbike.org/our-work/street-campaigns/the-wiggle/)** (shown below) that minimizes **hills (https://www.youtube.com/watch?v=u4fTf_qztns)** 🎥while also connecting popular biking routes.


Hayes St.
Baker St.
Fell St.
Divisadero St.
ALAMO SQUARE
Oak St.
Hickory St.
Webster St.
Lily St.
Koshland Park
Page St.
Rose St.
Pierce St.
Steiner St.
Haight St.
Broderick St.
Laussat St.
Buchanan St.
Lyon St.
Fillmore St.
Waller St.
San Francyclo
Wiggle Bicycles
Hermann St.
United States Mint
Lloyd St.
Duboce Ave.
California Volunteers Memorial
Buena Vista Park
Buena Vista Ter.
Alpine Ter.
Castro St.
Noe St.
Walter St.
Sanchez St.
Belcher St.
Park Hill Ave.
Market St.
14th St.
Churchill
Landers St.
Dolores St.
DUBOCE TRIANGLE
Cafe du Nord
Roosevelt Wy.

Bicycle networks are visible from low zoom levels. Networks include international, national, regional and local routes sourced from OpenStreetMap relations `bicycle_network: icn, ncn, rcn, lcn`.

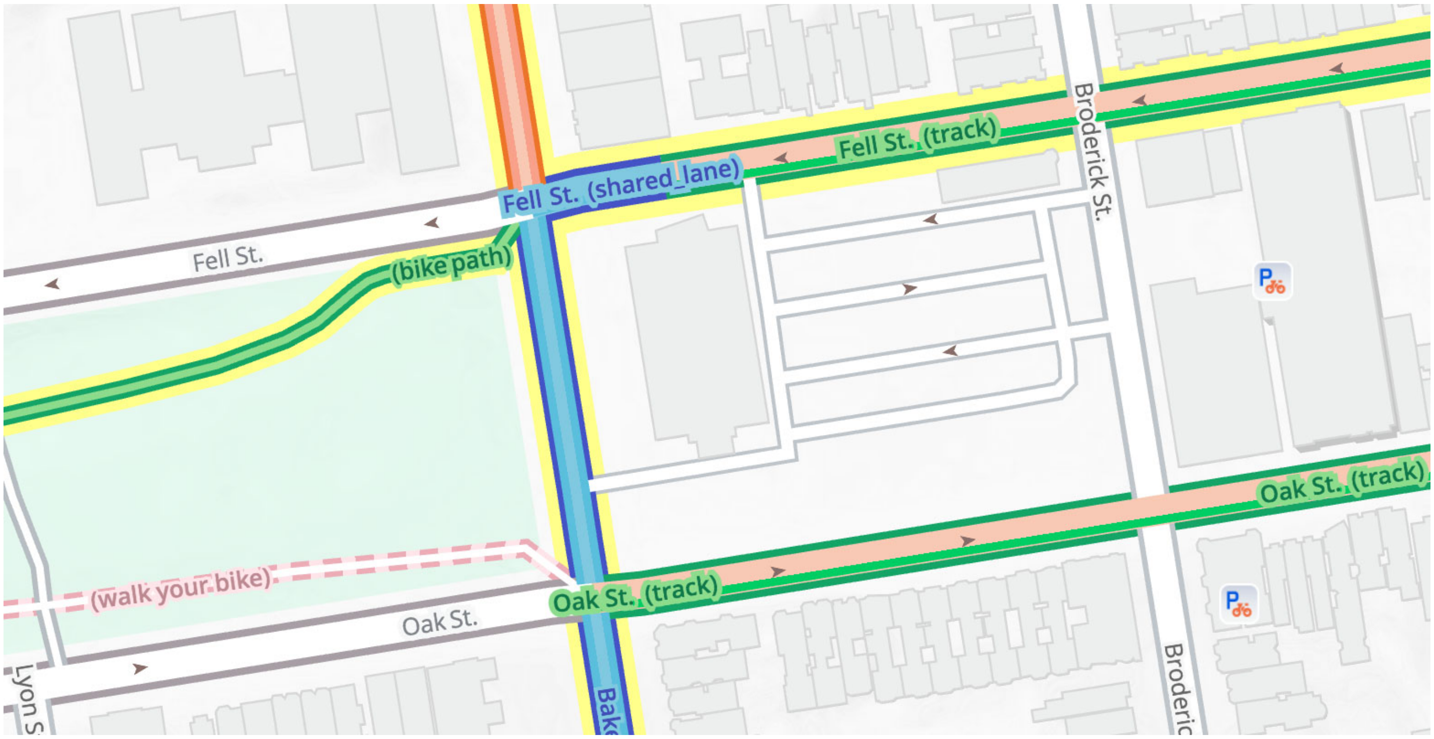

Mendocino
Willits
Mendocino Fores
Berryes Mountain Monu
Ukiah
101
Clear
101
Healdsburg
Santa Rosa
Petaluma
Point Reyes National Seashore
Richmond
San Francisco
80
Richmond
Concord
4
San Francisco
Oakland
San Bruno
Hayward
Fremont
280
1
San José
101
Boulder Creek
Santa Cruz
Watsonv
Salinas
1
Sole

Pacific Coast Bicycle Route in northern California, at zoom 8.

Bike Labels

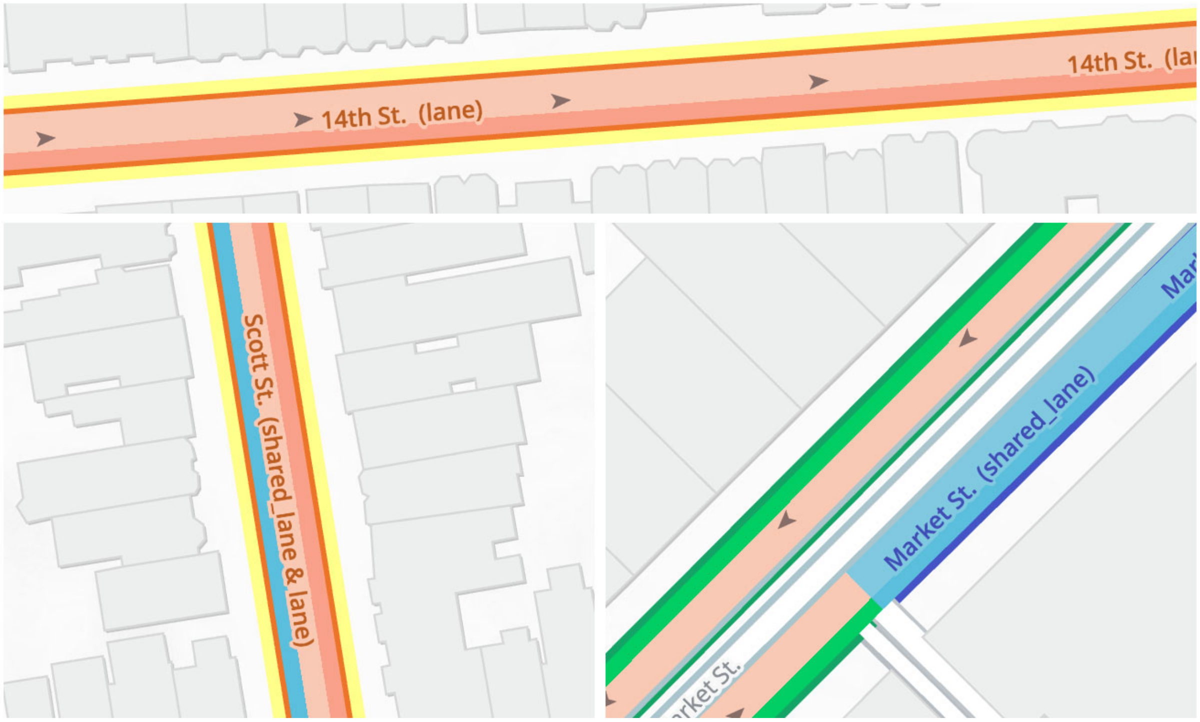
Bike labels are displayed starting at zoom level 18, along with right and left colored casings that indicate where to ride your bike.

Wondering what "green" means? Zoom into 18 and see that some are paths and some are cycle tracks along Fell and Oak Streets. Wondering what "blue" means? Zoom in and see "shared lane" along Baker St.


Fell St. (track)
Broderick St.
Fell St. (shared_lane)
Fell St.
(bike path)
(walk your bike)
Oak St.
Oak St. (track)
Oak St. (track)
Lyon S
Bake
Broderic

The “lane” label on 14th Street (below) indicates that there is a bike lane separated from car traffic, and orange inner line shows the lane is located on the right hand side of the road.

Sometimes things get complicated and different sides of the road have different types of bicycle facilities. Scott St. (below) has a “orange” bike lane going north but only a “blue” shared lane going south.

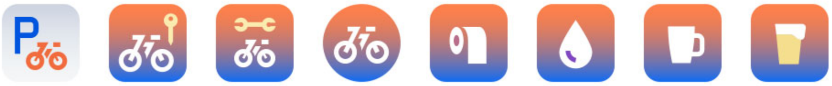

14th St. (lane)
14th St. (la
Scott St. (shared_lane & lane)
Mar
Market St. (shared_lane)
rket St.

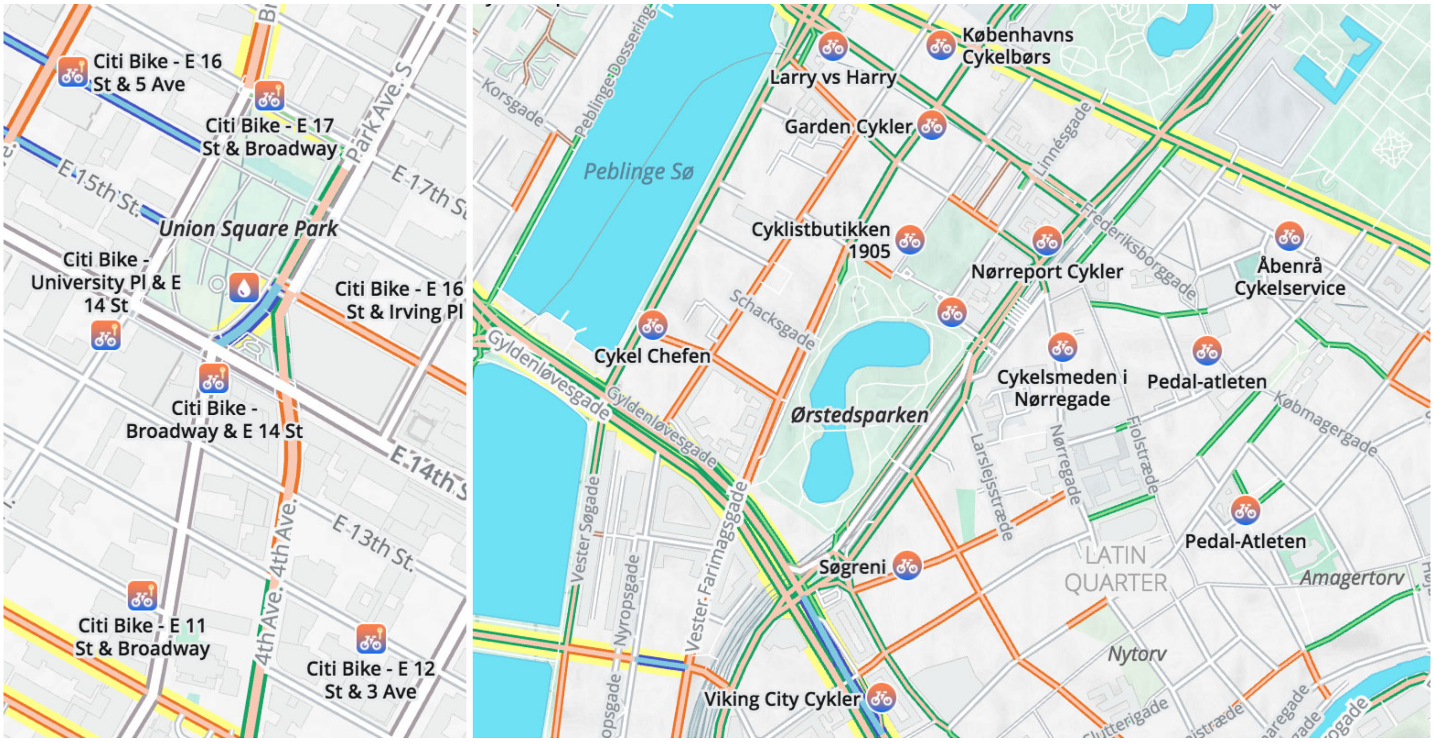
Special thanks to **Patricio Gonzalez Vivo (https://twitter.com/patriciogv)** and **Brett Camper (https://github.com/bcamper)** for shader help! See how to create this special left and right casing in Tangram:

```
casing_both:
    base: lines
    texcoords: true
    shaders:
        uniforms:
            OFFSET: 0.7
            blocks:
                color: |
                    vec2 st = v_texcoord.xy;
                    if (st.x < OFFSET && 1. - st.x < OFFSET) {
                        discard;
                    }
casing_left:
    mix: casing_right
    shaders:
        uniforms:
            FLIP: true
casing_right:
    base: lines
    texcoords: true
    shaders:
        uniforms:
            FLIP: false
            OFFSET: 0.7
        blocks:
            color: |
                vec2 st = v_texcoord.xy;
                if (FLIP) {
                    st.x = 1. - st.x;
                }

                if (st.x < OFFSET) {
                    discard;
                }
```

Bike Icons

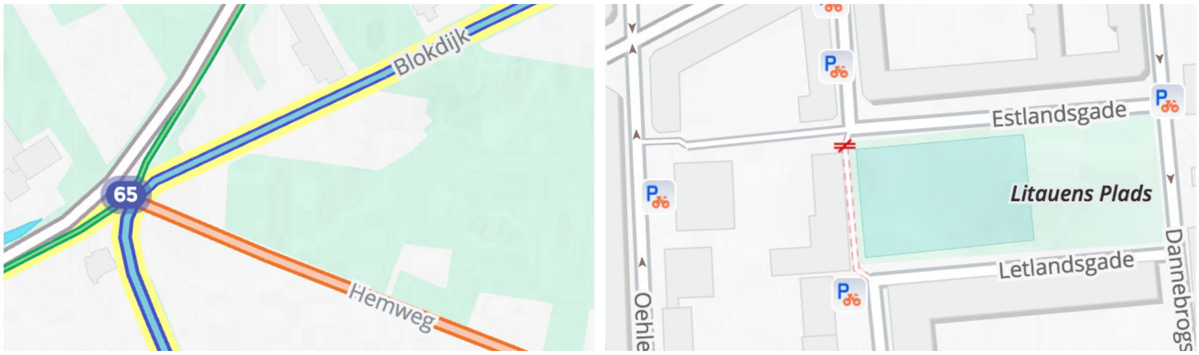
The bike overlay enables a special icon treatment for places related to bicycling, including: bicycle parking, bicycle rentals, bicycle repair stations, bicycle shops, drinking water, toilets, cafes, and pubs.




Citi Bike - E 16 St & 5 Ave
Citi Bike - E 17 St & Broadway
Park Ave. S
E 15th St.
E 17th St
Union Square Park
Citi Bike - University Pl & E 14 St
Citi Bike - E 16 St & Irving Pl
Citi Bike - Broadway & E 14 St
E 14th St
E 13th St.
4th Ave.
Citi Bike - E 11 St & Broadway
Citi Bike - E 12 St & 3 Ave
Korsgade
Peblinge Dossering
Peblinge Sø
Larry vs Harry
Københavns Cykelbørs
Garden Cykler
Linnésgade
Cyklistbutikken 1905
Nørreport Cykler
Frederiksborggade
Åbenrå Cykelservice
Schacksgade
Gyldenløvesgade
Cykel Chefen
Ørstedsparken
Cykelsmeden i Nørregade
Pedal-atleten
Købmagergade
Larslejsstræde
Nørregade
Fiolstræde
Vester Søgade
Nyropsgade
Vester Farimagsgade
Søgreni
LATIN QUARTER
Pedal-Atleten
Amagertorv
Nytorv
Viking City Cykler

New York and Copenhagen bike icons

We've also added bicycle junctions (for **cycle node networks (http://wiki.openstreetmap.org/wiki/Cycle_Node_Network_Tagging)** common in Europe) and cycle barriers.


Blokdijk
65
Hemweg
Estlandsgade
Litauens Plads
Letlandsgade
Oehle
Dannebrogs

Bicycle junction in the Netherlands and cycle barriers in Copenhagen

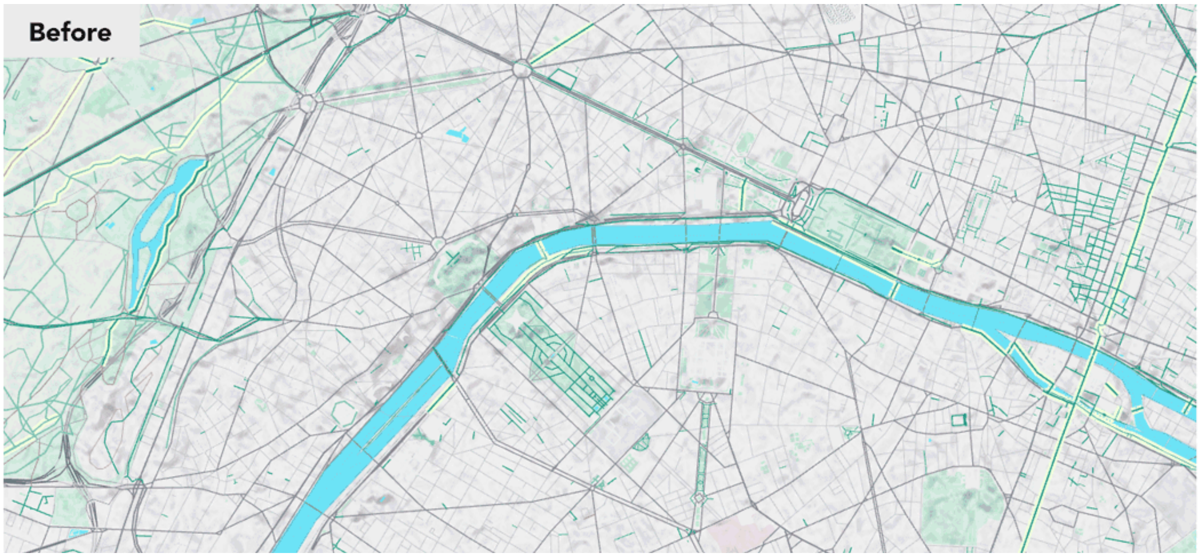
Walkabout Improvements

Roads Redesign

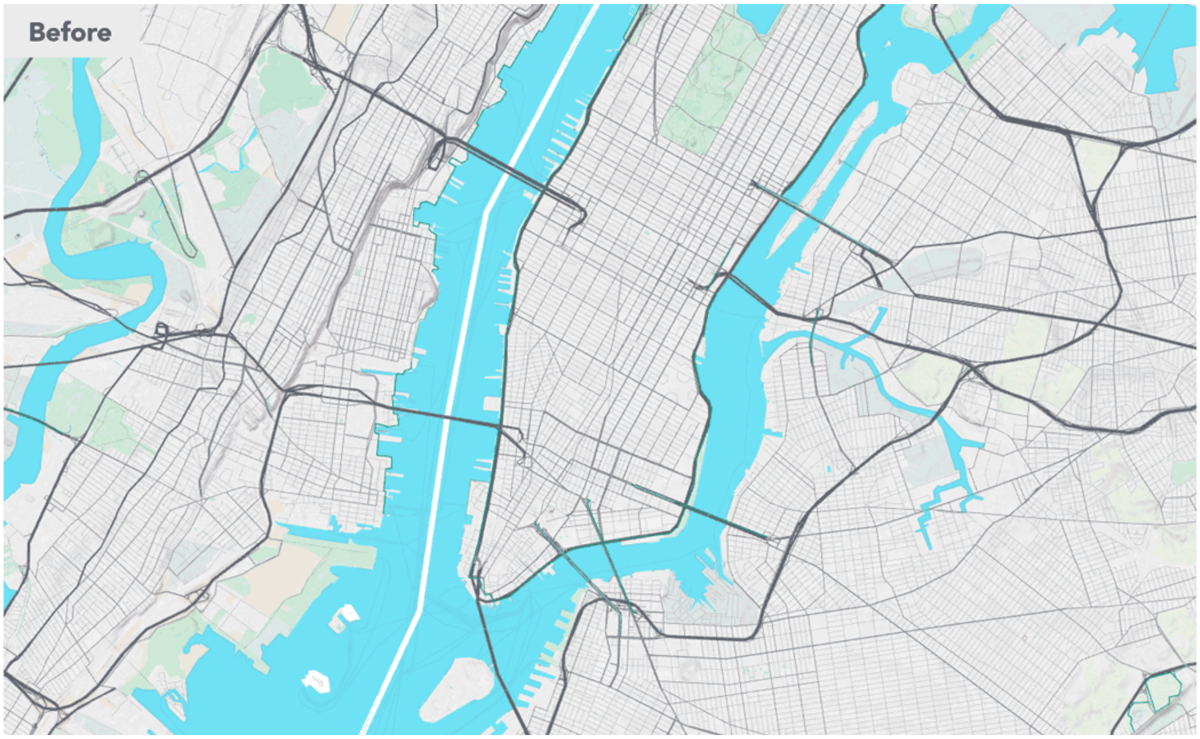
Before adding the new bike overlay, we reworked the way the roads are drawn in Walkabout. These changes have made a huge improvement on the overall color of the map and helps provide greater legibility.

The first improvement was on road colors and transitions. Before, with the use of the pencil spheremap and thin gray lines for roads, there was a tendency for the overall color of the map to feel “dirty” gray.

Our solution is to open up highways, primary and secondary roads at lower zoom levels and start minor roads as white instead of gray when they first appear at zoom 12. See Paris at zoom level 13 before and after comparison below.

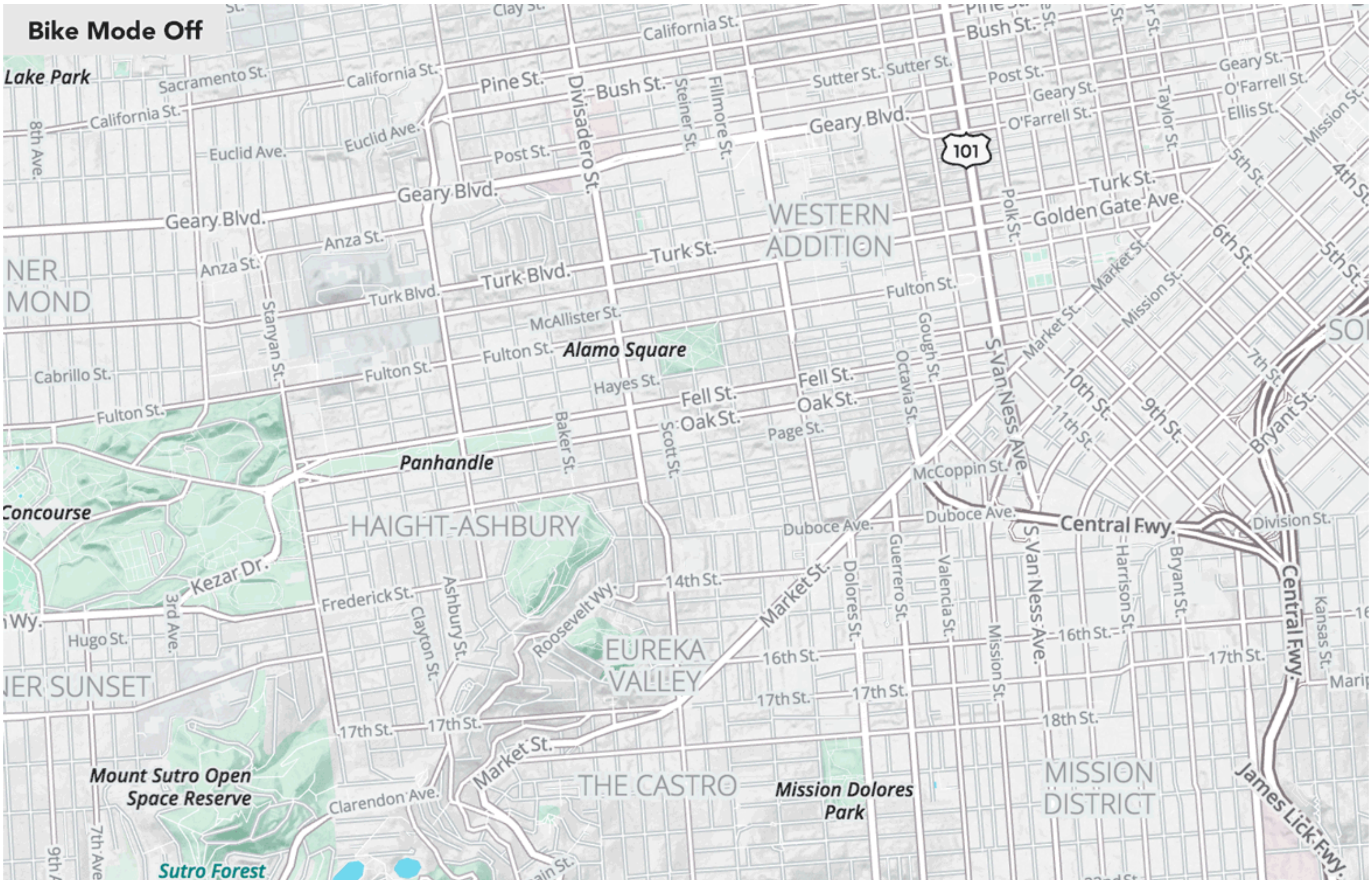


Here’s a comparison view of New York at zoom level 12.



All the road changes were in anticipation of the special way we've integrated the bike overlay in the yaml. Instead of adding the bike information on top of the basemap, we've weaved the various bike categories into the roads structure itself.

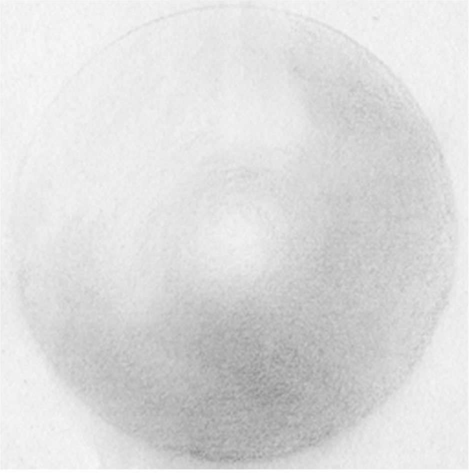
There is no second "overlay" layer drawn on top. Since we know from the vector tiles if a road is bike related, we can modify the appearance of the road as it's drawn the *first* time. This way, it keeps its width, but adjusts the color according to the tiers described above (and adds the inner line casings as appropriate). Nathaniel and I started calling this the "bike interlay", but the SDK continues the overlay pattern from transit.


Bike Mode Off
Lake Park
Sacramento St.
California St.
Pine St.
Bush St.
Divisadero St.
Steiner St.
Fillmore St.
Sutter St.
Post St.
Geary St.
O'Farrell St.
Ellis St.
Taylor St.
Mission St.
Euclid Ave.
Geary Blvd.
101
Turk St.
Golden Gate Ave.
WESTERN ADDITION
Polk St.
Anza St.
Turk Blvd.
Stanyan St.
McAllister St.
Fulton St.
Alamo Square
Gough St.
Octavia St.
S Van Ness Ave.
Market St.
Cabrillo St.
Hayes St.
Fell St.
Oak St.
Page St.
10th St.
11th St.
9th St.
Bryant St.
Panhandle
Baker St.
Scott St.
McCoppin St.
Concourse
HAIGHT-ASHBURY
Duboce Ave.
Central Fwy.
Division St.
Kezar Dr.
Frederick St.
Ashbury St.
Clayton St.
Roosevelt Wy.
14th St.
Dolores St.
Guerrero St.
Valencia St.
Mission St.
Harrison St.
Kansas St.
3rd Ave.
Hugo St.
EUREKA VALLEY
16th St.
17th St.
18th St.
Mount Sutro Open Space Reserve
Clarendon Ave.
THE CASTRO
Mission Dolores Park
MISSION DISTRICT
James Lick Fwy.
Sutro Forest
7th Ave.
8th Ave.

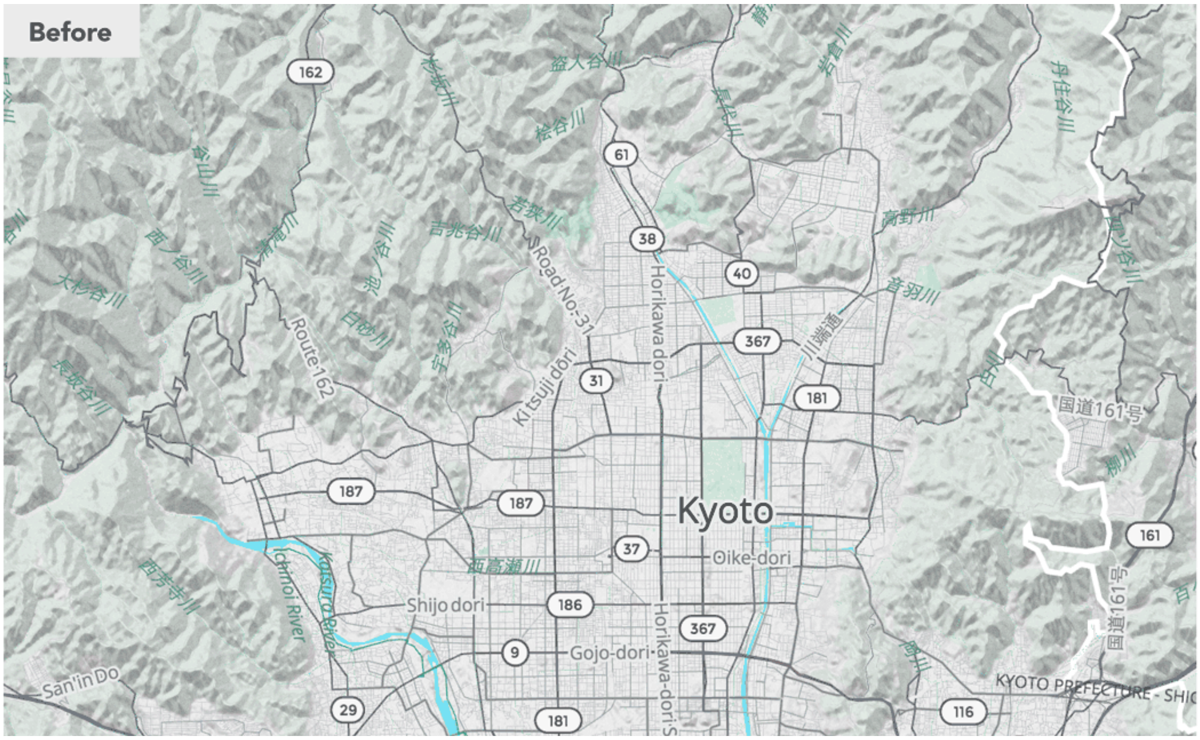
New Green Sphere Map

We now have an additional sphere map for green areas. In the initial version of Walkabout, color was added on top of gray. This also increased the tendency for the “dirty” gray overall color of the map.

With the addition of the new sphere map, we’re now mixing various shades of greens for all green areas. Along with this and other shader adjustments, the greens are more pure and there is more definition on the shading of terrain.

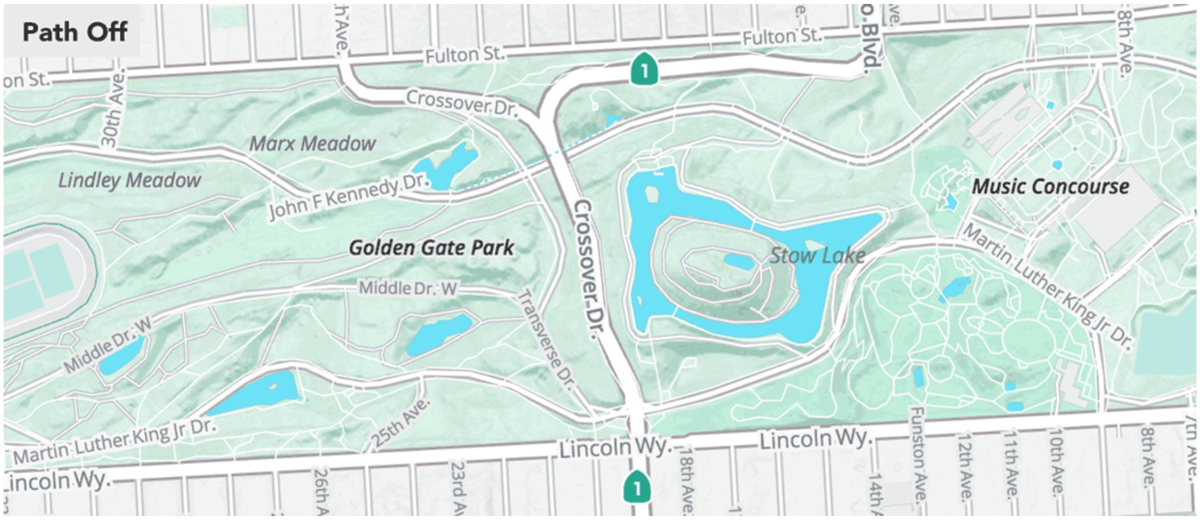


See a comparison view of Kyoto before and after below.


Before
Kyoto
Horikawa dori
Oike-dori
Shijo dori
Gojo-dori
Route 162
Road No.31
Kitsuji dori
Ichinoi River
Katsura River
San'in Do
国道161号
KYOTO PREFECTURE - SHIG

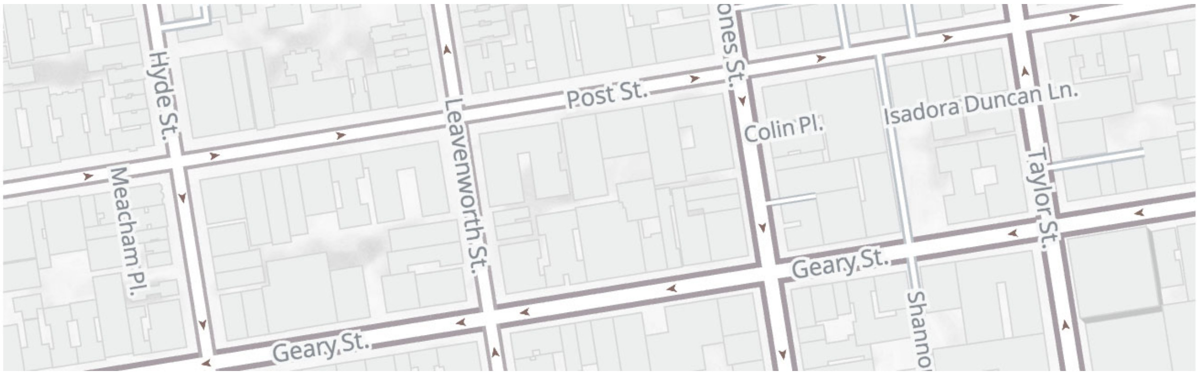
Walkabout: Now Trails Optional

We’ve also upgraded the paths and trails along with the new roads. Paths and trails are now an optional overlay that can be turned on and off as well. See Golden Gate Park below with and without the path overlay. This makes Walkabout suitable for a wider range of (not walking) applications.


Path Off
Fulton St.
Fulton St.
30th Ave.
Crossover Dr.
Marx Meadow
Lindley Meadow
John F Kennedy Dr.
Golden Gate Park
Crossover Dr.
Stow Lake
Music Concourse
Martin Luther King Jr Dr.
Middle Dr. W
Middle Dr. W
Transverse Dr.
Martin Luther King Jr Dr.
25th Ave.
Lincoln Wy.
Lincoln Wy.
Lincoln Wy.
26th A
23rd A
18th Ave.
14th A
Funston Ave.
12th Ave.
11th Ave.
10th Ave.
8th Ave.
8th Ave.
1
1

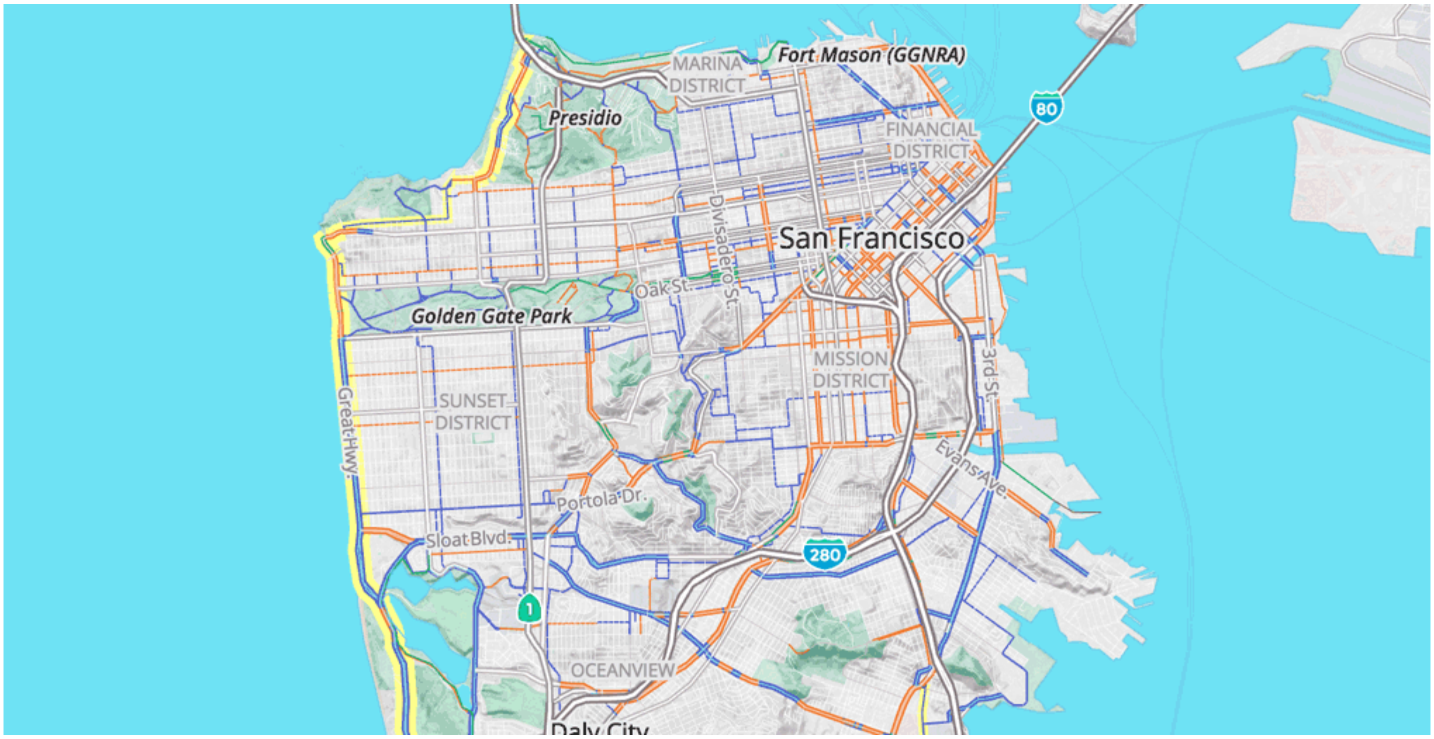
One-Way Arrows

Walkabout roads now include one-way arrows. Arrows start at zoom level 17 and are important for bicycling with the direction of car traffic (or in rare cases *opposite* for contraflow cycle lanes).


Hyde St.
Post St.
Leavenworth St.
ones St.
Colin Pl.
Isadora Duncan Ln.
Taylor St.
Meacham Pl.
Geary St.
Geary St.
Shanno

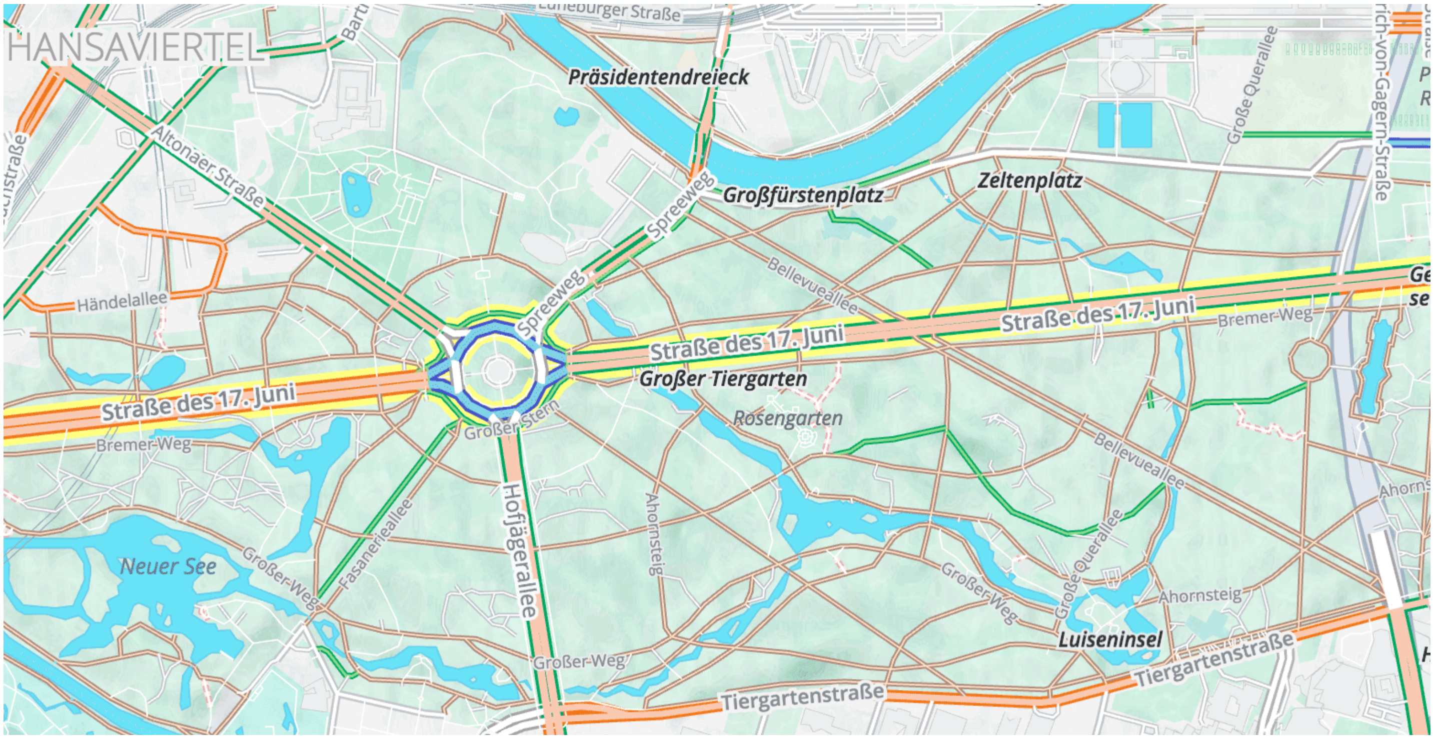
Gallery

See how bike-friendly your city is! Below are some zoom level 12 views of various cities. Notice all the green in Copenhagen, considered **number one bike capital (https://www.wired.com/2015/06/copenhagenize-worlds-most-bike-friendly-cities/)** of the world.

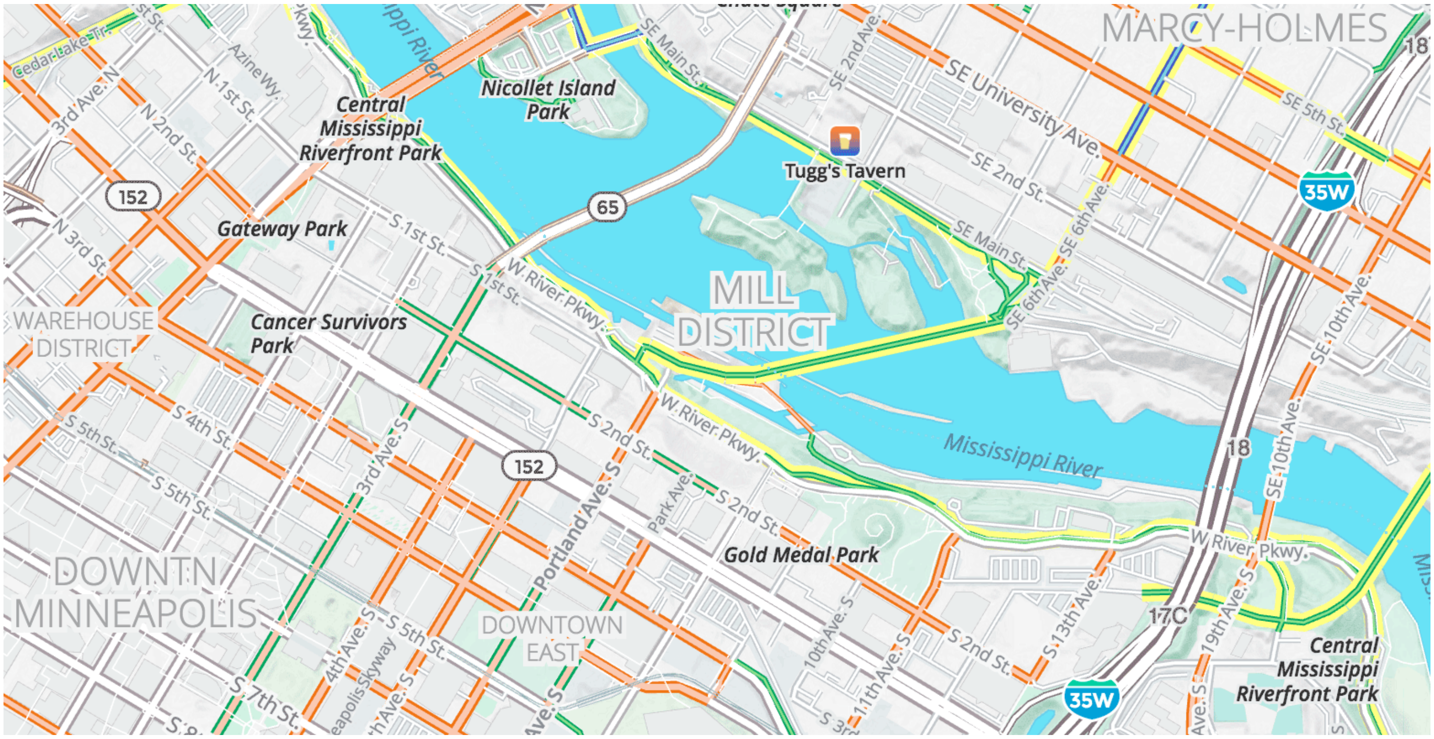

MARINA DISTRICT
Fort Mason (GGNRA)
Presidio
80
FINANCIAL DISTRICT
San Francisco
Divisadero St
Oak St
Golden Gate Park
MISSION DISTRICT
3rd St
SUNSET DISTRICT
Great Hwy
Evans Ave
Portola Dr
Sloat Blvd
280
1
OCEANVIEW
Daly City

Zoom 12 Bike map views for San Francisco, New York, Copenhagen, London, Berlin and Portland

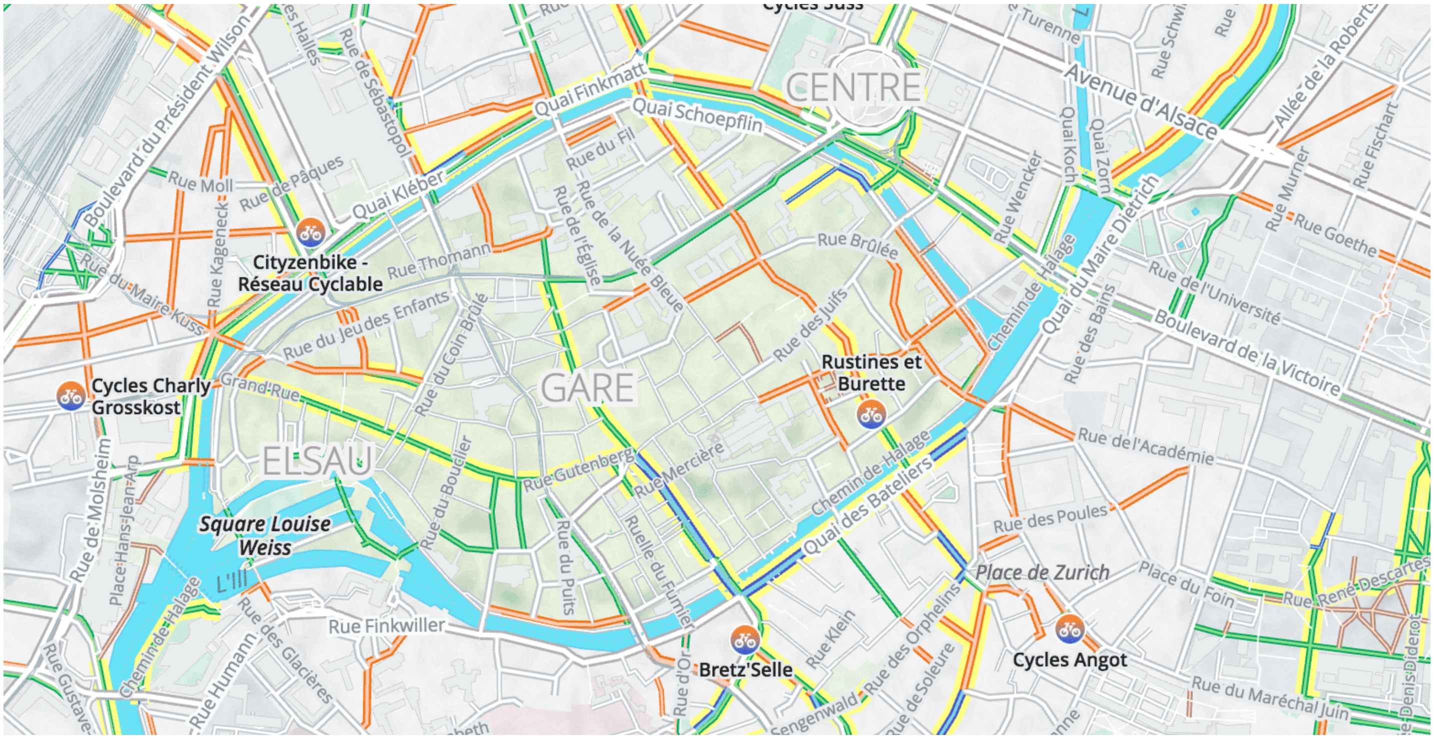
Here are some more views of bike-friendly cities:


HANSAVIERTEL
Lüneburger Straße
Präsidentendreieck
Altonaer Straße
Große Querallee
Spreeweg
Großfürstenplatz
Zeltenplatz
Händelallee
Bellevueallee
Straße des 17. Juni
Bremer Weg
Großer Tiergarten
Großer Stern
Rosengarten
Bellevueallee
Hofjägerallee
Fasanerieallee
Ahornsteig
Neuer See
Großer Weg
Große Querallee
Ahornsteig
Luiseninsel
Tiergartenstraße

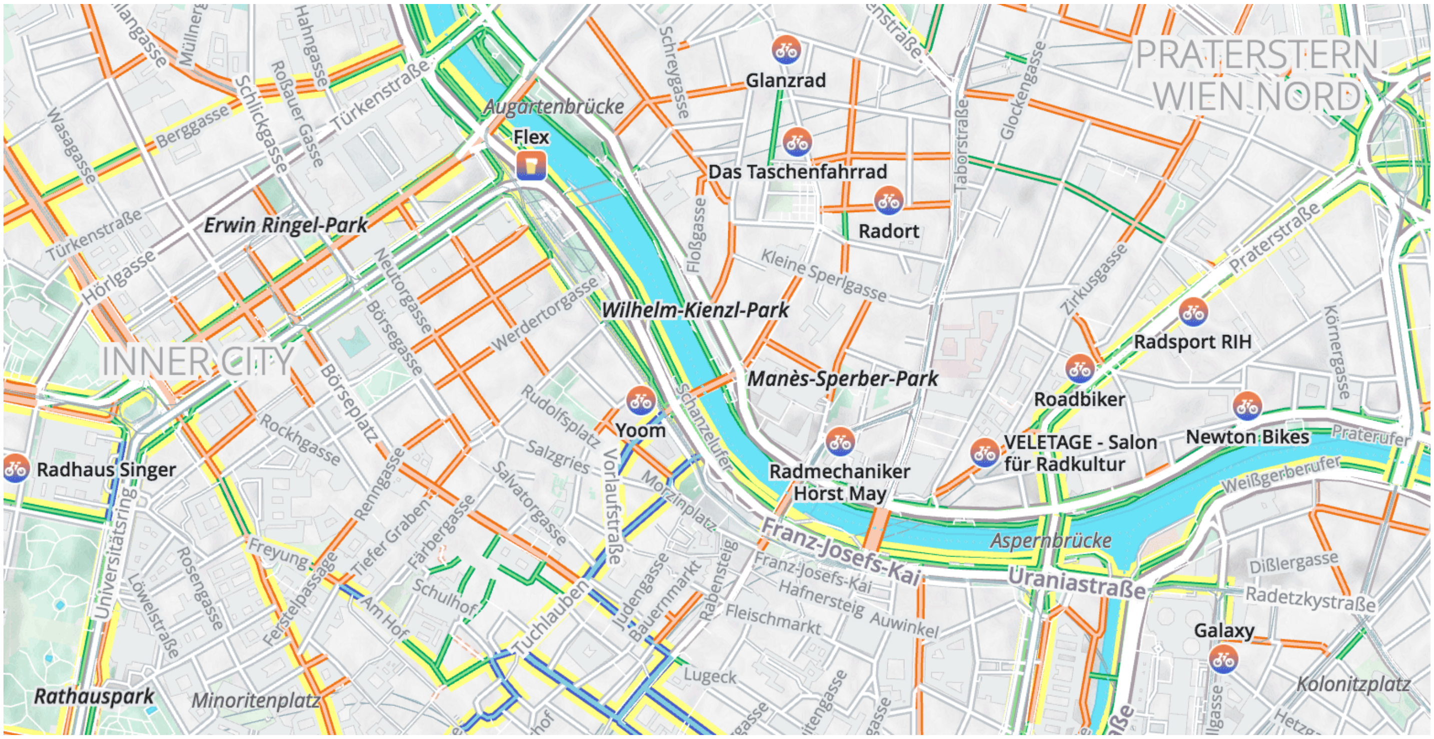
Berlin



Minneapolis



Strasbourg


PRATERSTERN WIEN NORD
INNER CITY
Glanzrad
Flex
Das Taschenfahrrad
Radort
Radsport RIH
Roadbiker
Newton Bikes
VELETAGE - Salon für Radkultur
Radmechaniker Horst May
Yoom
Radhaus Singer
Galaxy
Augartenbrücke
Erwin Ringel-Park
Wilhelm-Kienzl-Park
Manès-Sperber-Park
Aspernbrücke
Franz-Josefs-Kai
Uraniastraße
Rathauspark
Minoritenplatz
Kolonitzplatz
Praterstraße

Vienna

Powered by Mapzen Vector Tiles

Thanks to **Matt Amos (https://github.com/zerebubuth)** and **Rob Marianski (https://github.com/rmarianski)**, Mapzen Vector Tiles include **bicycle properties (https://mapzen.com/blog/v0.10-tiles/)** on road features directly, including data traversed from OpenStreetMap relations to highlight national and regional cycle networks.

Are you curious how to tag roads in your city on OpenStreetMap so they show up on Walkabout's bike overlay? Check out our **Cycleways Targeted Editing (https://mapzen.com/blog/targeted-editing-cycleways/)** blog post. Changes made in OpenStreetMap will show up in a few hours in Mapzen Vector Tiles.

This new version of Walkabout is the first to require a Mapzen API key to enable access to Mapzen vector tiles. Read more about the change in our **earlier post (https://mapzen.com/blog/basemap-style-update/)**, and get a free API key **here (https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys)**.

Besides updating the map, changes in OpenStreetMap will also update Mapzen Turn-by-Turn directions to improve our **elevation influenced (https://mapzen.com/blog/making-the-grade-elevation-influenced-bicycle-routing/) bicycle routing (https://mapzen.com/blog/valhalla-bicycle-routing-options/)**.

Get the Bike Overlay

Just like **enabling languages (https://mapzen.com/blog/languages-of-india/)**, it's very easy to turn on the bike overlay after importing the Walkabout basemap style in Tangram – just import the basemap and add a few lines! The Cartography **docs (https://mapzen.com/documentation/cartography/api-reference/#bike-overlay)** include the following example and bonus map legend.

Set the required **Mapzen API key (https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys)** to access vector and terrain tiles, disable the path overlay (which is on by default), and enable the optional bike overlay:

```
import: https://mapzen.com/carto/walkabout-style/4/walkabout-style.zip

global:
  sdk_mapzen_api_key: mapzen-xxxxxx          # your API key here
  sdk_path_overlay: false
  sdk_bike_overlay: true
```

If you're using **mapzen.js (https://github.com/mapzen/mapzen.js)**:

```
var map = L.Mapzen.map('map', {
  center: [40.8041, -124.1506],
  zoom: 15,
  maxZoom: 20,
  tangramOptions: {
    scene: {
      import: L.Mapzen.BasemapStyles.Walkabout,
      global: {
        sdk_mapzen_api_key: 'mapzen-xxxxxx',  // your API key here
        sdk_path_overlay: false,
        sdk_bike_overlay: true
      }
    }
  }
});
```

Let us know (mailto:hello@mapzen.com&subject=%22bike%20map%22) what you think. We are working on a colorblind-safe mode and would like your feedback.

Also, it turns out that people have… opinions about map colors. Our choices are admittedly qualitative, but we wanted a “warm” set of colors that would let cyclists readily see safer routes and would work well with the Walkabout palette.

The good news it that if you are so opinionated, you are not stuck with our colors as you would be with raster tiles! The magic of vector tiles and Tangram lets you readily change colors on the fly via global variables – **try it in Tangram Play (https://mapzen.com/tangram/play/?scene=https://mapzen.com/api/scenes/22/331/resources/blank.yaml#17.03753/37.76927/-122.44174)**!

Just uncomment (command-slash in Tangram Play!) the bike tier you want to change, **using hex, rgb, hsl or even human-friendly HTML/CSS color names (https://mapzen.com/documentation/tangram/draw/#color)**.

- `_color` is the casing/outline.
- `_color_casing` is for bike lanes on the left and right side of the road at zoom 18.
- `_color_lite` is the interior of the road.

```
import: https://mapzen.com/carto/walkabout-style/4/walkabout-style.zip

global:
    sdk_mapzen_api_key: mapzen-xxxxxx          # your API key here
    sdk_path_overlay: false
    sdk_bike_overlay: true
    # bike_tier1_off_road_path_color: [0.085,0.650,0.405] # green casing / outline
    # bike_tier1_off_road_path_color_casing: [0.000,0.810,0.392] # green left-right lanes
    # bike_tier1_off_road_path_color_lite: # [0.554,0.865,0.554] # green fill
    # bike_tier1_off_road_footway_color: [0.690,0.547,0.504]  # light brown tracks (unpav
    # bike_tier1_off_road_footway_color_lite: [1.000,0.881,0.770] # lighter brown tracks
    # bike_tier1_off_road_track_color: [0.500,0.343,0.290]  # brown casing / outline
    # bike_tier1_off_road_track_color_lite: [0.917,0.759,0.610] # brown fill
    # bike_tier1a_label: [0.063,0.474,0.294] #green label
    # bike_tier1b_label: [0.559,0.338,0.269] #brown label
    # bike_tier2_on_road_protected_color: [0.930,0.460,0.167] # orange casing
    # bike_tier2_on_road_protected_color_casing: [0.980,0.635,0.545] # orange left-right
    # bike_tier2_on_road_protected_color_lite: [0.980,0.792,0.706] # orange fill
    # bike_tier2_label: [0.743,0.367,0.135] # orange label
    # bike_tier3_on_road_not_protected_color: [0.273,0.332,0.780] #[0.361,0.416,0.800] #
    # bike_tier3_on_road_not_protected_color_casing: [0.357,0.750,0.870] # blue left-righ
    # bike_tier3_on_road_not_protected_color_lite: [0.502,0.792,0.880] #[0.660,0.829,0.88
    # bike_tier3_label: [0.253,0.307,0.719] # blue label
    # bike_prohibited_color: [0.930,0.679,0.721] # pink dashed line (no bikes/walk your b
    # bike_prohibited_color_lite: [1.000,1.000,1.000] # pink fill
    # # regional_network_highlight: [0.973,1.000,0.318] # yellow highlight
    # regional_network_highlight: [1.000,1.000,0.549] # yellow hightlight
    # no_bike_label: [0.740,0.540,0.572] # pink casing
    # no_bike_color_lite: [1.000,0.898,0.918] # pink fill
```

(Make sure you **add your API key (https://mapzen.com/developers)** on line 4 to see your changes or your map will be sad. Also, the `/4/` in the `import` url is important – we may change these color variable names in the future.)

Safe biking!

· 09 March 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

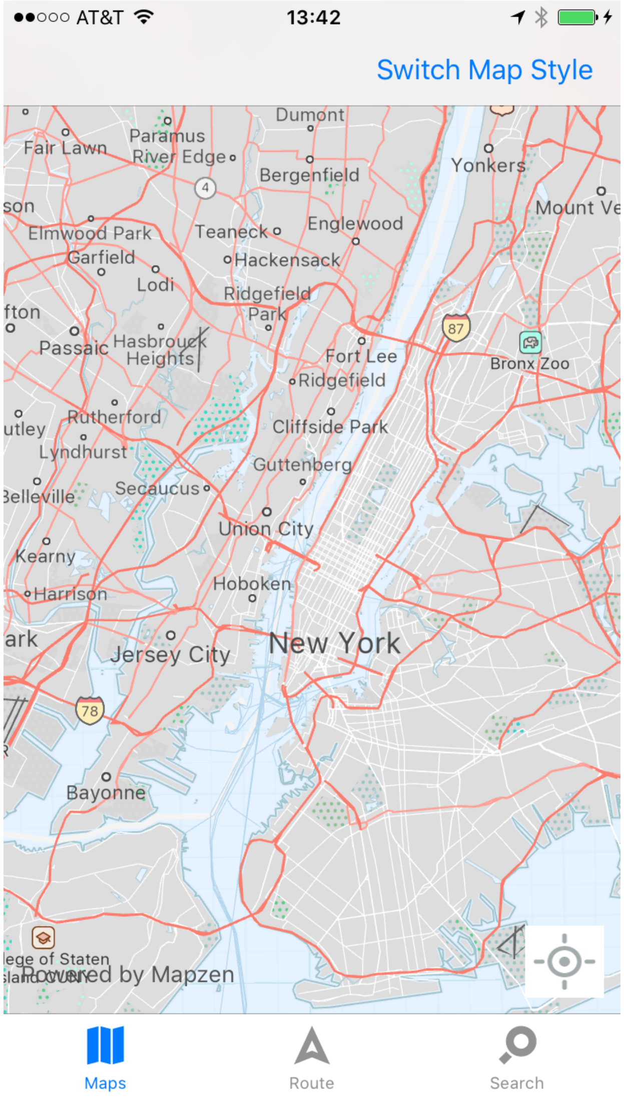
Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

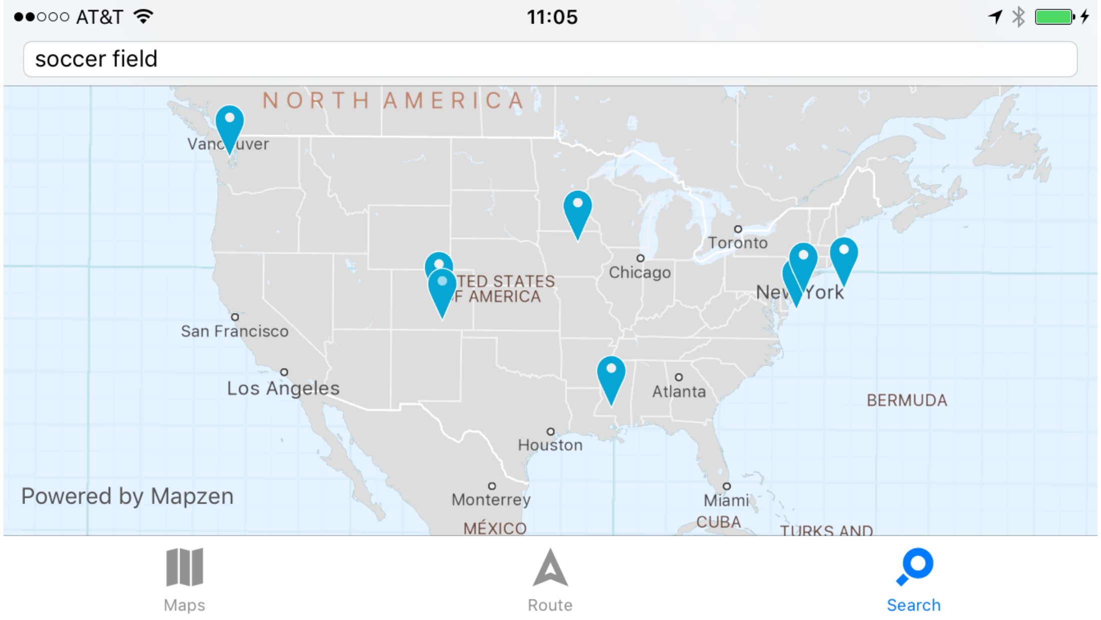
© 2017 Mapzen

Mapzen iOS SDK is now in Beta

mobile (/tag/mobile)

The iOS SDK Beta!


●●○○○ AT&T
13:42
Switch Map Style
Dumont
Paramus
Fair Lawn
River Edge
Bergenfield
Yonkers
Mount Ve
son
Elmwood Park
Teaneck
Englewood
Garfield
Hackensack
Lodi
Ridgefield Park
fton
Passaic
Hasbrouck Heights
Fort Lee
Bronx Zoo
Ridgefield
Rutherford
utley
Cliffside Park
Lyndhurst
Guttenberg
Belleville
Secaucus
Union City
Kearny
Hoboken
Harrison
ark
Jersey City
New York
Bayonne
lege of Staten Island CUNY
Powered by Mapzen
Maps
Route
Search


●●○○○ AT&T
11:05
soccer field
NORTH AMERICA
Vancouver
Toronto
Chicago
UNITED STATES OF AMERICA
New York
San Francisco
Los Angeles
Atlanta
BERMUDA
Houston
Powered by Mapzen
Monterrey
MÉXICO
Miami
CUBA
TURKS AND
Maps
Route
Search

If you have been following along **on GitHub (https://github.com/mapzen/ios)**, then you probably have seen a recent flurry of activity as we push towards the release of our long-awaited Mapzen iOS SDK.

Part of that work is the recent beta release to **CocoaPods (https://cocoapods.org/pods/Mapzen-ios-sdk)**, which we feel is ready for community testing!

What's Included

Version 1.0 of our iOS SDK will be nearly at feature parity with our well established **Android SDK (https://github.com/mapzen/android)**.

Major features include:

- High performance and highly customizable map rendering using OpenGL ES provided by **Tangram-es (https://github.com/tangrams/tangram-es)**.
- Driving directions and customizable route lines provided by **Mapzen Turn-by-Turn (https://mapzen.com/products/turn-by-turn/)**.
- Geocoding and Point-of-Interest search provided by **Mapzen Search (https://mapzen.com/products/search/)**.
- Several **base map styles (https://mapzen.com/products/maps/)** to suit most use cases.

And many more features than we can list here in a timely fashion.

You can browse the complete API documentation for the beta release **here (https://mapzen.github.io/ios/)**.

How Do I Get The Beta?

Step 1: Get yourself a free **Mapzen API Key (https://mapzen.com/developers/sign_up)**.

Step 2: Install the beta SDK through the **CocoaPods (https://cocoapods.org/pods/Mapzen-ios-sdk)**.

Step 3: Check out the sample app **source code (https://github.com/mapzen/ios)** or `pod try Mapzen-ios-sdk` to load it immediately. You'll need that API key from step 1 in either case. See where to set it **here (https://github.com/mapzen/ios#configure-api-key)**.

Step 4: Let us know your thoughts! You can either open a new **issue on GitHub (https://github.com/mapzen/ios/issues)** or send us email at **ios-support@mapzen.com (mailto:ios-support@mapzen.com)**.

· 13 March 2017 ·

Matt Smollinger

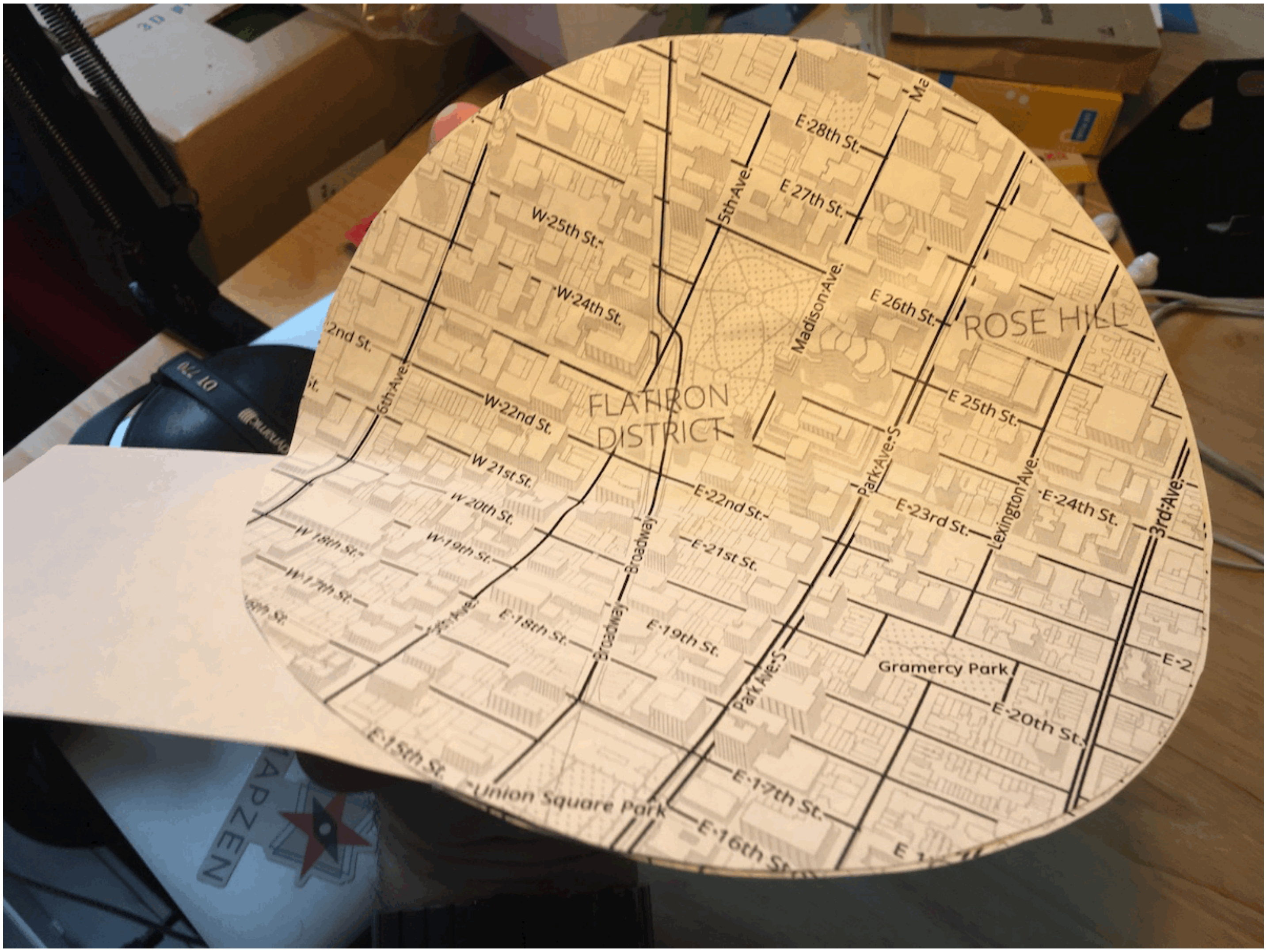
Matt would 3D print an SDK if it were possible.

© 2017 Mapzen

NYC Office Hours (and tacos)

New York City, your time has come! Tomorrow from 11 am to 1 pm, Mapzen Office Hours meets Manhattan. **Register now (https://goo.gl/forms/z0WEUNmuXOotKLnb2)**, and come on by to ask questions (and get answers!) about Mapzen tools and maps. We'll also be showing people how to use **mapzen.js (https://mapzen.com/documentation/mapzen-js/)**!

Also… did we mention that we'll have tacos for lunch? **Map sandwiches (https://mapzen.com/blog/map-sandwich/)**, meet map tacos!


E 28th St.
E 27th St.
5th Ave.
W 25th St.
W 24th St.
Madison Ave.
E 26th St.
ROSE HILL
2nd St.
6th Ave.
W 22nd St.
FLATIRON DISTRICT
E 25th St.
W 21st St.
W 20th St.
E 22nd St.
Park Ave S
E 23rd St.
Lexington Ave.
E 24th St.
3rd Ave.
W 18th St.
W 19th St.
Broadway
E 21st St.
W 17th St.
E 18th St.
E 19th St.
Gramercy Park
Park Ave S
E 20th St.
E 15th St.
Union Square Park
E 17th St.
E 16th St.
MAPZEN

Sign up now (https://goo.gl/forms/z0WEUNmuXOotKLnb2) and swing on by!

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

open this taco map fullscreen (https://mapzen.com/tangram/view/?scene=https://mapzen.com/api/scenes/22/402/resources/scene.yaml#17.05417/40.74407/-73.99108)

· 21 March 2017 ·

Mapzen

Opening maps with open source and open data.

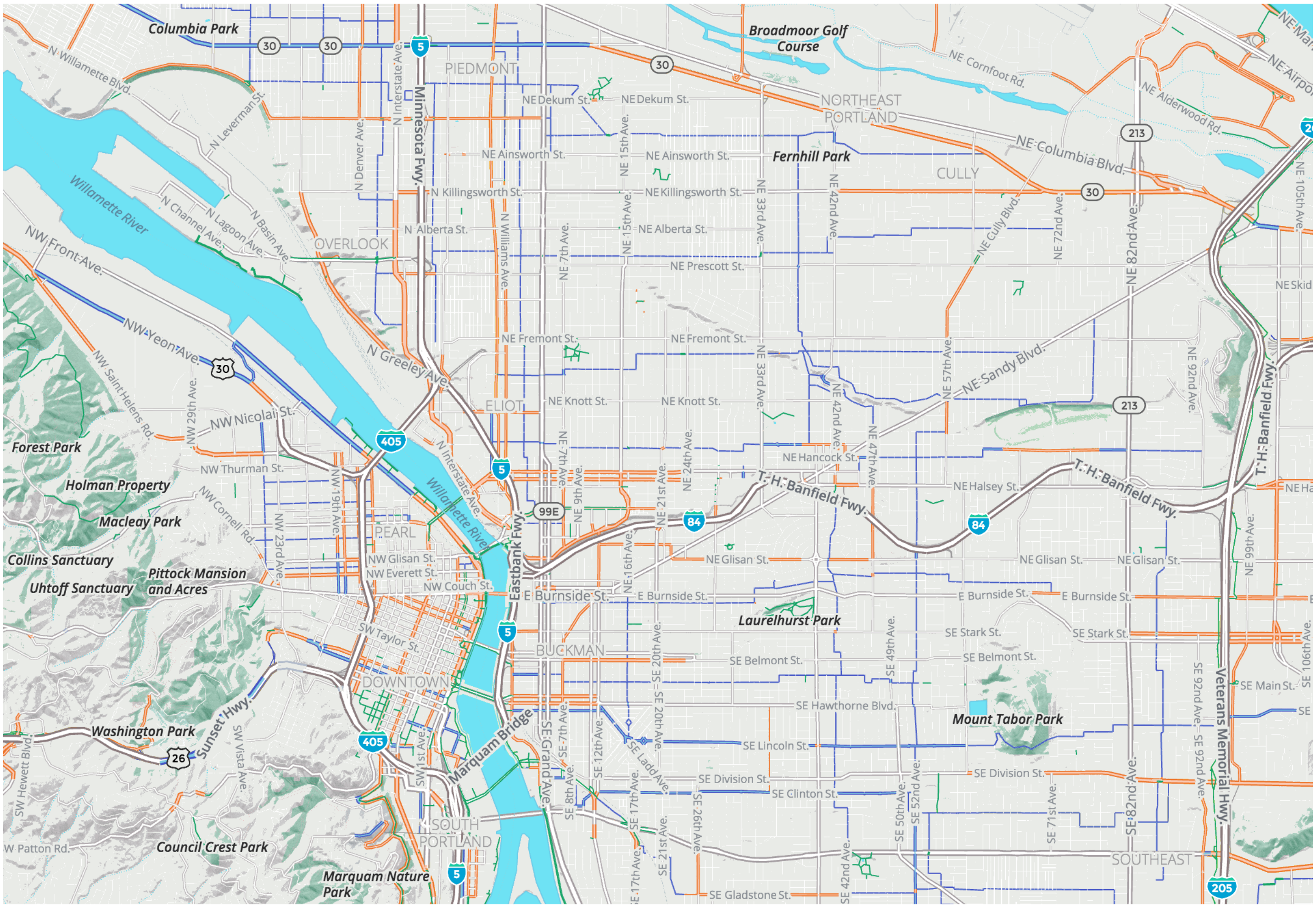
© 2017 Mapzen

Bike maps and you

tangram (/tag/tangram) cartography (/tag/cartography)

We heard your comments and updated the bike map, thank you! **Checkout the new version... (http://mapzen.com/blog/bike-map-v2)**

As you may have heard, **we made a bike map (https://mapzen.com/blog/bike-map/)**! We've gotten a lot of great feedback and are working on making safer routes more prominent on the map, combining bike route relations with bike infrastructure and street type. Here's a sneak peek of how our bike map is evolving in Portland:


Columbia Park
N Willamette Blvd.
PIEDMONT
Minnesota Fwy.
N Interstate Ave.
NE Dekum St.
Broadmoor Golf Course
NE Cornfoot Rd.
NORTHEAST PORTLAND
NE Alderwood Rd.
NE Columbia Blvd.
NE Airport
N Leverman St.
N Denver Ave.
NE Ainsworth St.
NE 15th Ave.
Fernhill Park
CULLY
Willamette River
N Channel Ave.
N Lagoon Ave.
N Basin Ave.
N Killingsworth St.
NE Killingsworth St.
NE 33rd Ave.
NE 42nd Ave.
NE Cully Blvd.
NE 72nd Ave.
NE 82nd Ave.
NE 105th Ave.
NW Front Ave.
OVERLOOK
N Alberta St.
N Williams Ave.
NE 7th Ave.
NE Alberta St.
NE Prescott St.
NE Skid
NW Yeon Ave.
N Greeley Ave.
NE Fremont St.
NE 57th Ave.
NE Sandy Blvd.
NE 92nd Ave.
T. H. Banfield Fwy.
NW Saint Helens Rd.
NW 29th Ave.
ELIOT
NE Knott St.
NW Nicolai St.
Forest Park
N Interstate Ave.
NE 24th Ave.
NE Hancock St.
NE 47th Ave.
NW Thurman St.
Holman Property
NE Halsey St.
NE 9th Ave.
NE 21st Ave.
NW 19th Ave.
Macleay Park
NW Cornell Rd.
NW 23rd Ave.
PEARL
Eastbank Fwy.
NE 16th Ave.
NE Glisan St.
NE 99th Ave.
Collins Sanctuary
Pittock Mansion and Acres
NW Glisan St.
NW Everett St.
NW Couch St.
Uhtoff Sanctuary
E Burnside St.
Laurelhurst Park
NE 49th Ave.
SE Stark St.
SW Taylor St.
BUCKMAN
SE 20th Ave.
SE Belmont St.
SE 106th Ave.
DOWNTOWN
SE Main St.
SE 92nd Ave.
Veterans Memorial Hwy.
SE Hawthorne Blvd.
Washington Park
Sunset Hwy.
SW Vista Ave.
SW 1st Ave.
Marquam Bridge
SE Grand Ave.
SE 7th Ave.
SE 12th Ave.
SE Ladd Ave.
SE Lincoln St.
Mount Tabor Park
SW Hewett Blvd.
SE 8th Ave.
SE 17th Ave.
SE Division St.
SE 26th Ave.
SE Clinton St.
SE 50th Ave.
SE 52nd Ave.
SE 71st Ave.
SE 82nd Ave.
SOUTH PORTLAND
SE 21st Ave.
SE 42nd Ave.
SOUTHEAST
W Patton Rd.
Council Crest Park
Marquam Nature Park
SE Gladstone St.
5
30
405
84
99E
26
213
205

But we need your help to make it even better all across the planet! Bicycle infrastructure varies dramatically around the world, both in quality and name. Some places have bike-friendly, traffic-calmed Greenways and Bike Boulevards, while others have Green Waves with lights timed for bikes. Some places even have “living streets” where cars are permitted, but bikes and pedestrians have priority.

Please take this poll (https://docs.google.com/forms/d/1pAi1hWj8tW7ssxo9lOjW1xtcRYo3RUswvVQSmTxbkqg) and let us know more about bikes and safety in your city, as well as feedback on our map!

· 23 March 2017 ·

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

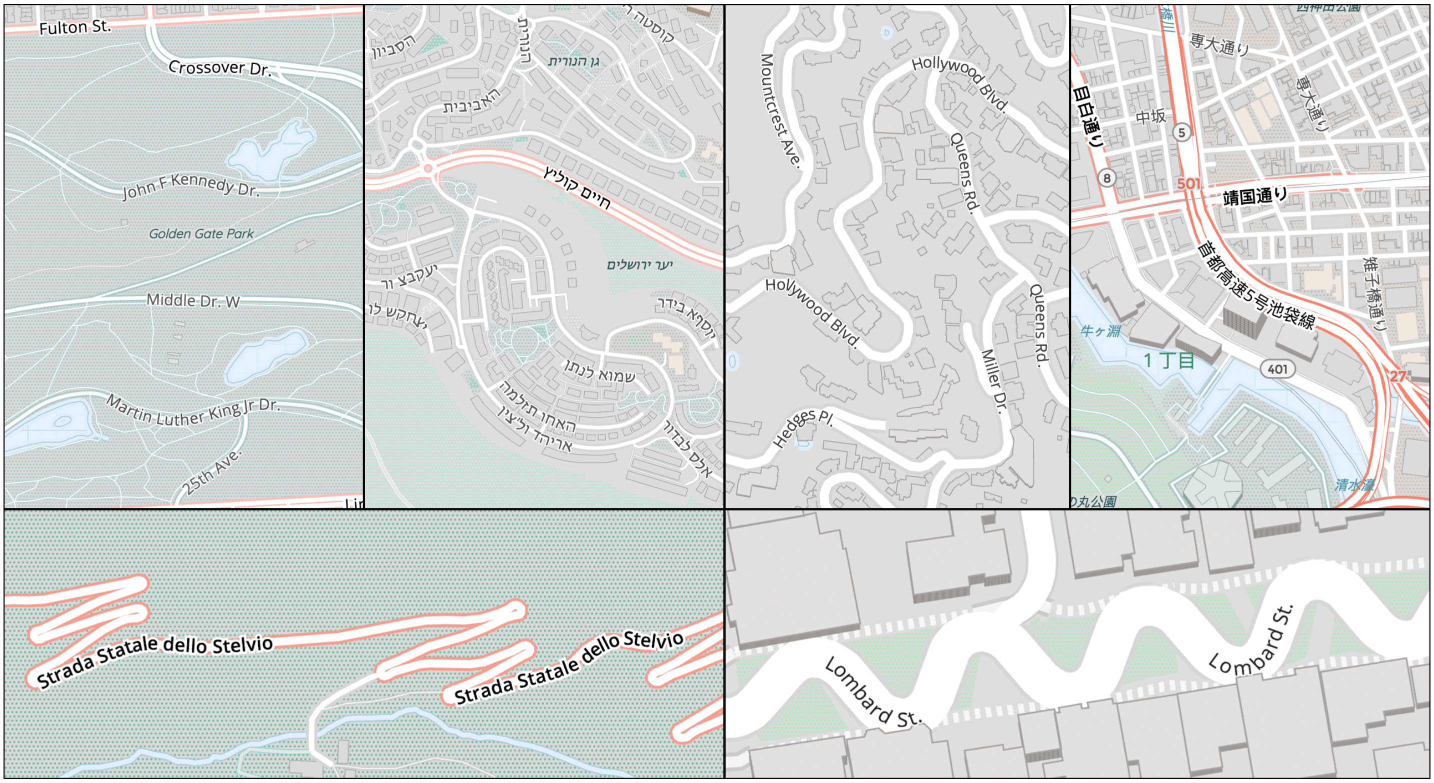
© 2017 Mapzen

The Twists and Turns of Curved Labels

tangram (/tag/tangram)

“We turn the cube and it twists us.” — Ernő Rubik

Tangram JS v0.12 (https://github.com/tangrams/tangram/releases/tag/v0.12.0) introduces curved labels, a feature that allows labels to curve along a road and smoothly animate when zoomed.


Fulton St.
Crossover Dr.
John F Kennedy Dr.
Golden Gate Park
Middle Dr. W
Martin Luther King Jr Dr.
25th Ave.
גן הנורית
יער ירושלים
Mountcrest Ave.
Hollywood Blvd.
Queens Rd.
Hollywood Blvd.
Queens Rd.
Miller Dr.
Hedges Pl.
専大通り
目白通り
中坂
靖国通り
首都高速5号池袋線
牛ヶ淵
1丁目
401
清水濠
Strada Statale dello Stelvio
Strada Statale dello Stelvio
Lombard St.
Lombard St.

In this post we do a technical deep dive into the somewhat unusual way this feature was implemented.

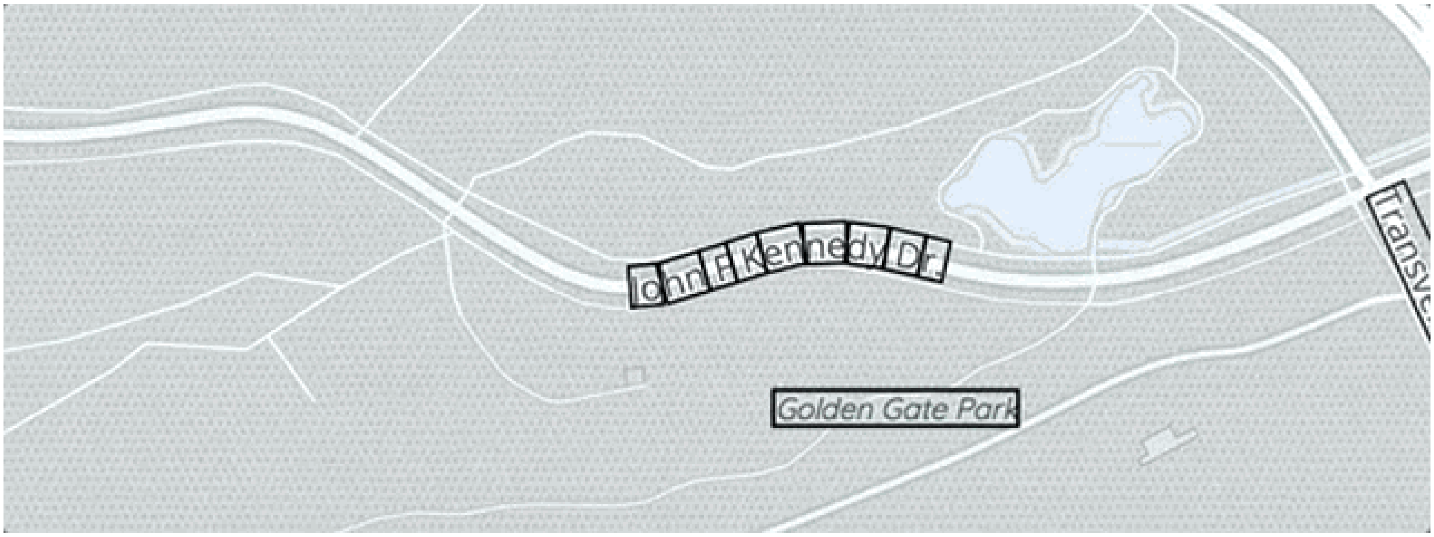
Technical Deep Dive

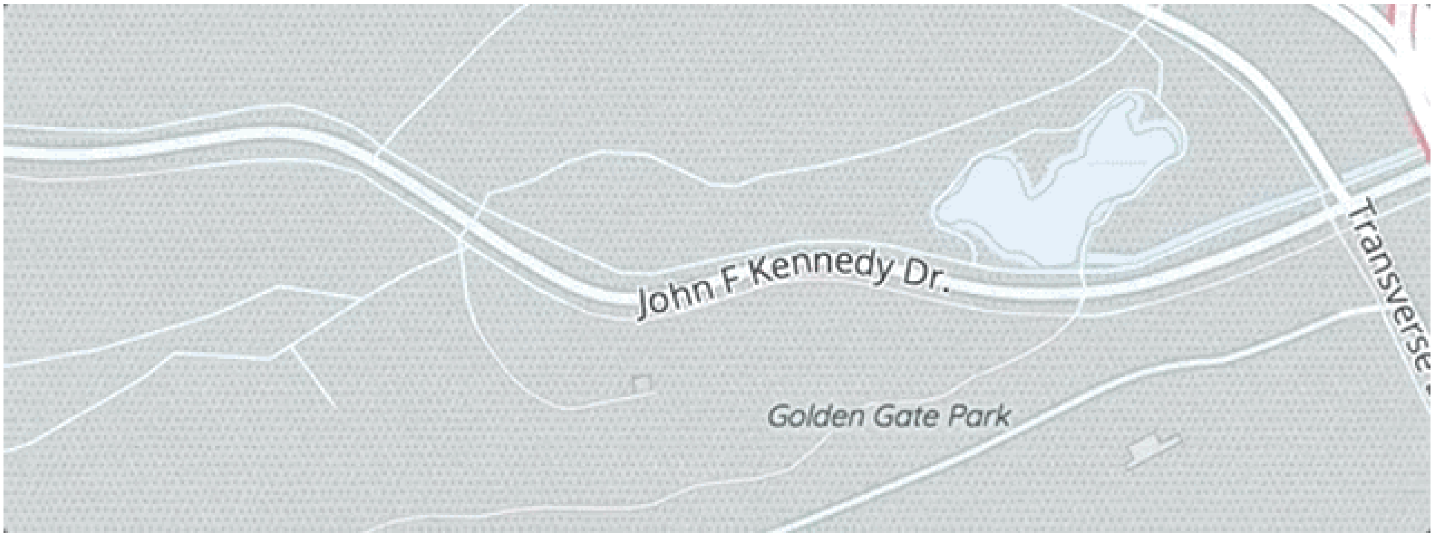
CURLY, STRAIGHT.
CURLY, STRAIGHT.

***via GIPHY and simpsonsworld.com* (https://giphy.com/gifs/season-5-the-simpsons-5x3-xT5LME7lsJdVu2C48g)**

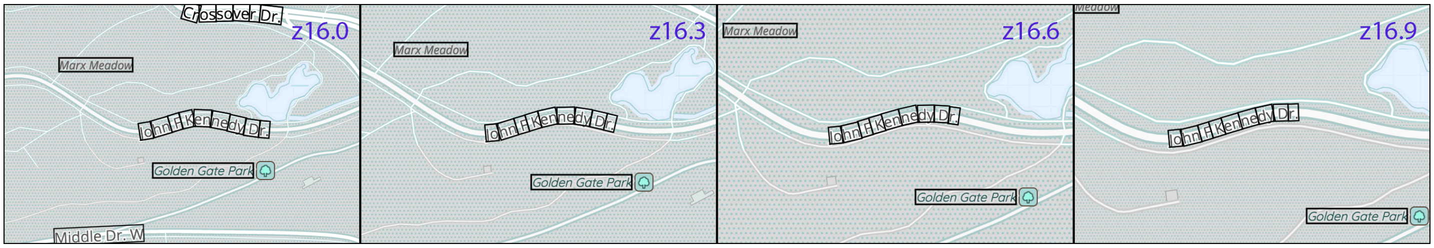
For labels to bend, they must first be broken up into pieces and reassembled along a curve. Each piece must be measured and placed along the line segments of the underlying geometry.

When the user zooms the map, the text stays the same size, but the road's size will scale. Since the angle and position of the text needs to follow the road, they will have to animate. This creates the effect of the text stretching along the road as it shrinks or expands.


John F Kennedy Dr.
Golden Gate Park
Transve


John F Kennedy Dr.
Transverse
Golden Gate Park

Recomputing the angles and positions of all the labels as the user scrolls is expensive and interrupts the fluid scrolling experience, so in Tangram JS, we take a shortcut. We figure out where the labels would be at four different zoom levels: 0, 0.33, 0.66, and 0.99 from the base zoom level.


Crossover Dr.
z16.0
Marx Meadow
John F Kennedy Dr.
Golden Gate Park
Middle Dr. W
z16.3
Marx Meadow
John F Kennedy Dr.
Golden Gate Park
z16.6
Marx Meadow
John F Kennedy Dr.
Golden Gate Park
Meadow
z16.9
John F Kennedy Dr.
Golden Gate Park

We then interpolate between these values in the vertex shader to get an approximate value for any zoom. The naive way to do this would be to use a bunch of `if` statements to detect which zoom levels we are in between. This isn't very performant though, as shader code **doesn't like conditionals (https://www.opengl.org/discussion_boards/showthread.php/181028-conditionals-in-shaders)**. We do something a little more clever, perhaps too clever. With a combination of `mix` and `step` functions, you can create the same logic as a conditional. When you have to do this with cascading conditionals that have `mix's` in them for interpolation themselves… things get complicated.

Here you'll see the shader code we use. The graph shows the interpolation between four values (these can be used for the angles, or the x- and y- positions) defined in the variables `a`, `b`, `c`, `d`. The values of the graph are draggable, so you can see how the function behaves as you modify them.

Language Support

"If it bends, it's funny. If it breaks, it's not funny." — Woody Allen

Whenever you break text up and put it back together you must consider the language it comes from. For right-to-left (RTL) languages like Arabic and Hebrew, you may be putting the text back together **backwards (https://mapzen.com/blog/backwards-and-compatible/)**! Moreover, many languages have **text-shaping (https://en.wikipedia.org/wiki/Complex_text_layout)** where a character is represented differently depending on the characters surrounding it. For example:

- In Greek, the σ (sigma) character is written differently if it occurs at the end of the word, as in σοφός (sofos)

- In Devanagari, the city of Taxila, Pakistan is spelled टैक्सिला, but the individual characters are क् + स + क्स.
- The Arabic character م **(https://en.wiktionary.org/wiki/%D9%85)** can look like “ـم” or “ـمـ” or “مـ” depending on how it connects to surrounding characters.
- In Hindi, अशांत means “not quiet”, but अश + अंत means passage + the end.

These are a few of many examples of how complicated scripts can be, and how thinking of characters as atomic units stripped of their context can lead to wildly inaccurate results. For the interested reader, **this (https://r12a.github.io/scripts/tutorial/part3)** is a great overview.

For languages with text-shaping we are forced to disable the label curving feature (for now). These languages are Arabic, Bengali, Burmese, Devanagari, Khmer, Gujarati, Gurmukhi, Kannada, Lao, Mongolian, Oriya, Tamil, Telugu and Tibetan. We do support RTL languages that don’t have text-shaping, such as Hebrew. In the research leading up to this feature, we’ve relied heavily on the work of Richard Ishida of the W3C such as **this handy chart (http://r12a.github.io/scripts/featurelist/)**.

JavaScript vs Native Implementations

> “In theory there’s no difference between practice and theory. In practice there is.” — Yogi Berra

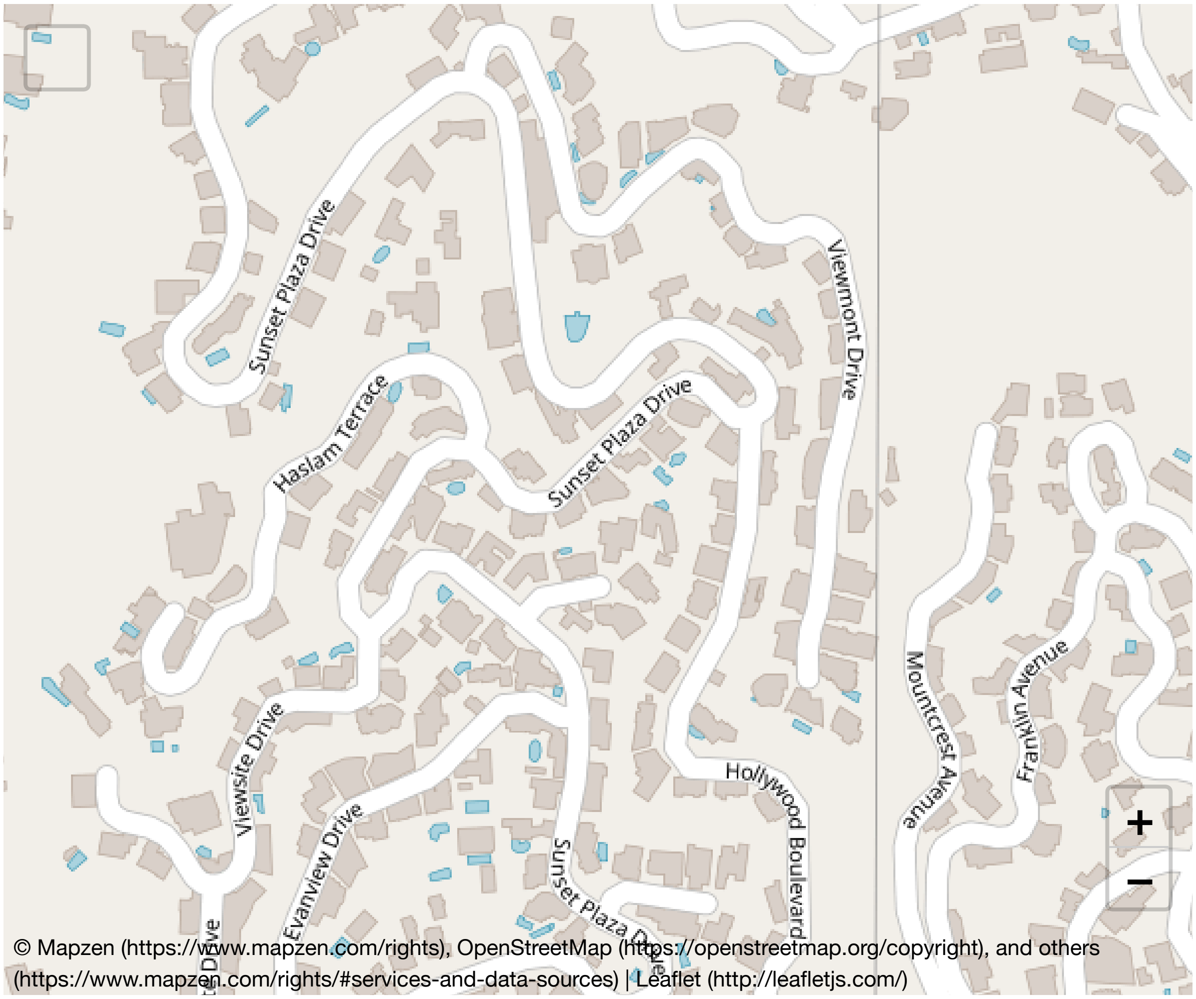
So far we’ve been detailing the JavaScript implementation for curved labels. Tangram ES, our native implementation for iOS, Android and other platforms, takes a different route for the same feature. This stems from important platform differences between web and native.

For the web, we measure widths of text using the `canvas.measureText API` and draw compound strings to an off-screen canvas element which gets converted to a texture. It’s by leveraging the canvas element’s built-in text rendering that enables TangramJS to render and measure text in any language and typographic style that canvas supports (which is a lot). We can load fonts from any font file at runtime and use the canvas built-in stroking and filling API to style text.

On the native side, Tangram ES comes bundled with **Harfbuzz (https://www.freedesktop.org/wiki/Software/HarfBuzz/)**, a sophisticated text layout engine written in C++. The API of Harfbuzz is much more versatile than what the web’s canvas can do. On the JS side, we could transpile Harfbuzz with emscripten, however that would lead to a much larger bundle size and longer intial load times.

In Tangram ES, each character of a string (whether straight or curved) is its own glyph (set of pixels referenced in a global texture). The ordering and placement of these glyphs is solved using Harfbuzz. As the user zooms, these values are recomputed on each tick. We can do that on Tangram ES because Harfbuzz is very fast and gives more granular information about the text string. In Tangram JS, however, measuring sizes is slower, and so we do it as infrequently as possible. Straight labels are drawn as a single texture, instead of being broken up by character. When no straight label can be placed, then we try to curve a label and break it up into pieces. We also do all the computation on initial tile load, so that if a user zooms, everything has been precomputed (as described above).

Check out your favorite curved streets around the world!


Sunset Plaza Drive
Viewmont Drive
Haslam Terrace
Sunset Plaza Drive
Viewsite Drive
Evanview Drive
Sunset Plaza Drive
Hollywood Boulevard
Mountcrest Avenue
Franklin Avenue
+
−
© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

· 30 March 2017 ·

David Valdman

Working on Tangram. Henceforth, it is the map that precedes the territory.

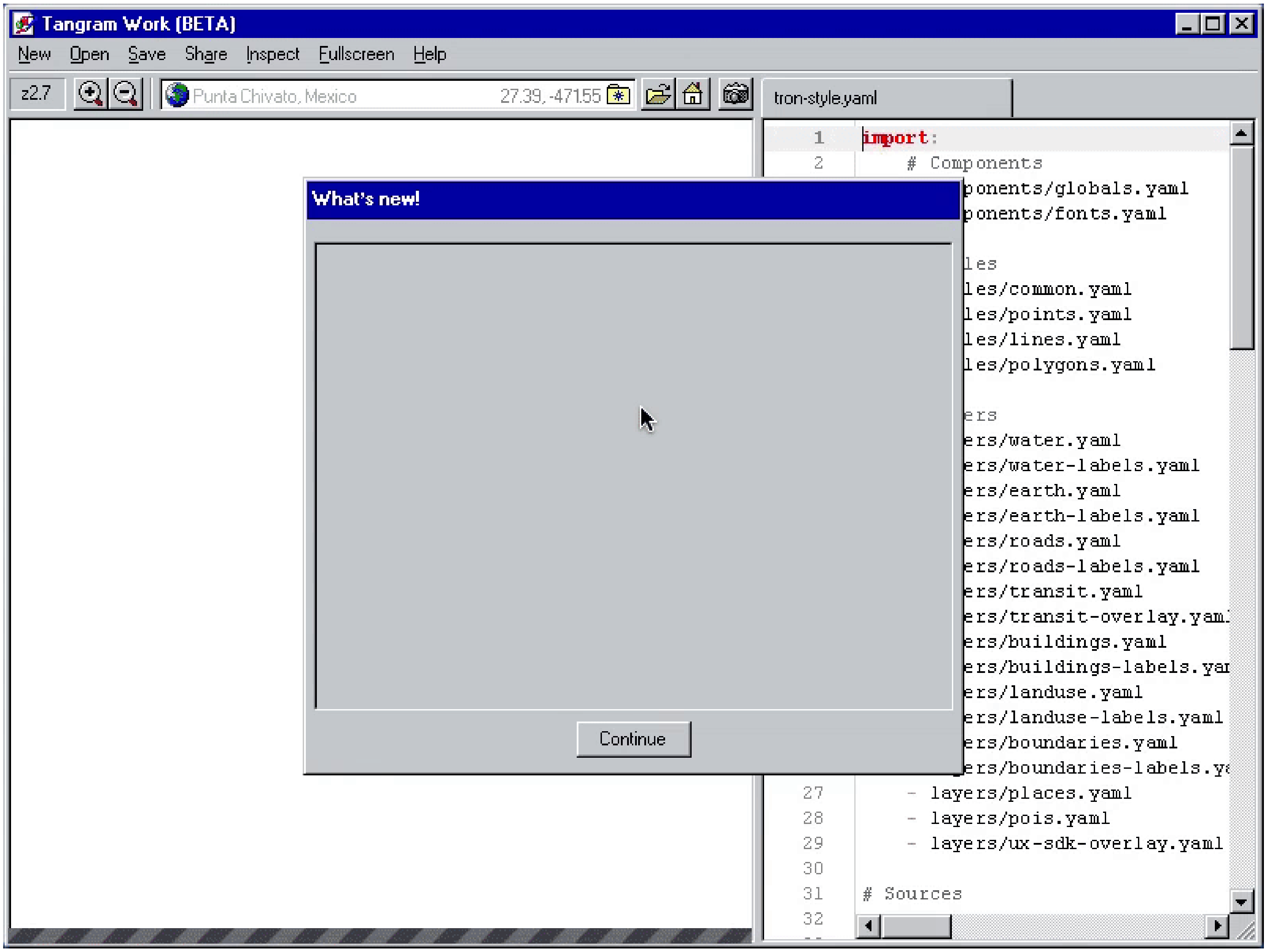
© 2017 Mapzen

Introducing Tangram Work

tangram (/tag/tangram)

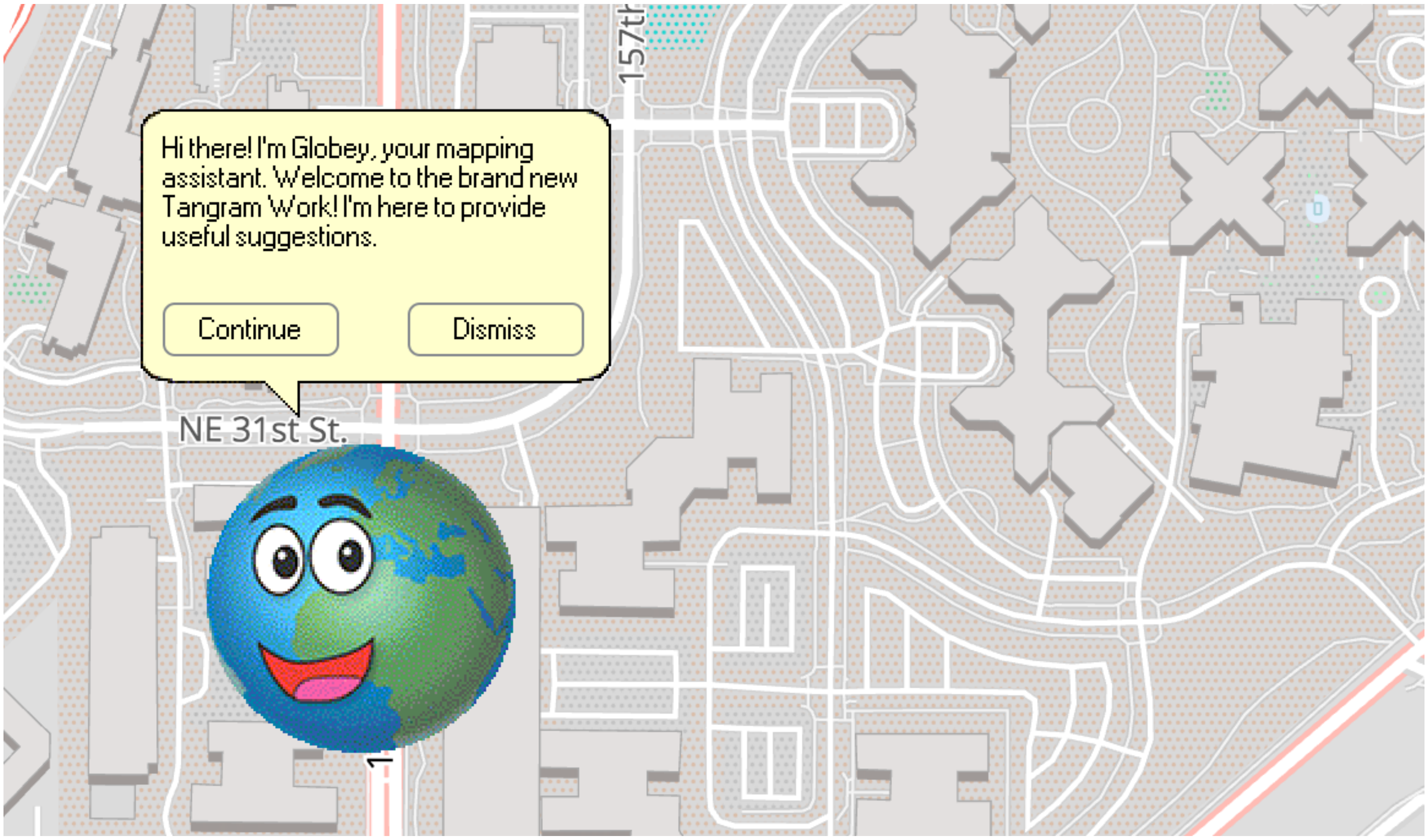
It's been almost two months since we **launched (https://mapzen.com/blog/tangram-play-public-beta-launch/)** our Tangram scene editor, **Tangram Play (https://mapzen.com/tangram/play/)**. We've learned a lot from our users during this time. One thing we've heard loud and clear is that our web-based interface did not meet the needs of our power users. So, starting today, we are rebranding Tangram Play to Tangram Work, designed completely for the enterprise user in mind.

Tangram Work has been completely redesigned to look and feel as professional and powerful as possible, based on decades of user research and cutting edge interface design. Much of the same functionality from Tangram Play has been ported directly to Tangram Work, but now you can look like you are doing real work instead of messing around on the internet.


Tangram Work (BETA)
New
Open
Save
Share
Inspect
Fullscreen
Help
z2.7
Punta Chivato, Mexico
27.39, -471.55
tron-style.yaml
What's new!
Continue

```
import:
    # Components
              ponents/globals.yaml
              ponents/fonts.yaml

              les
              les/common.yaml
              les/points.yaml
              les/lines.yaml
              les/polygons.yaml

              ers
              ers/water.yaml
              ers/water-labels.yaml
              ers/earth.yaml
              ers/earth-labels.yaml
              ers/roads.yaml
              ers/roads-labels.yaml
              ers/transit.yaml
              ers/transit-overlay.yam
              ers/buildings.yaml
              ers/buildings-labels.ya
              ers/landuse.yaml
              ers/landuse-labels.yaml
              ers/boundaries.yaml
              ers/boundaries-labels.y
    - layers/places.yaml
    - layers/pois.yaml
    - layers/ux-sdk-overlay.yaml

# Sources
```

We’re also incredibly proud to introduce Globey, our new, friendly productivity helper who will be your personal mapping assistant as you build new scenes with Tangram. Be sure to give Globey a try today!


157th
Hi there! I'm Globey, your mapping assistant. Welcome to the brand new Tangram Work! I'm here to provide useful suggestions.
Continue
Dismiss
NE 31st St.

With this pivot to Tangram Work, we won't immediately remove Tangram Play. However, you can use Tangram Work immediately! Tangram Work is a downloadable desktop application that can run on **Mac OS X (https://mapzen-assets.s3.amazonaws.com/resources/tangram-work/tangram-work-darwin-x64.zip)**, **Windows (https://mapzen-assets.s3.amazonaws.com/resources/tangram-work/tangram-work-win32-x64.zip)**, or **Linux (https://mapzen-assets.s3.amazonaws.com/resources/tangram-work/tangram-work-linux-x64.zip)** platforms.

Download now for Mac OS X

(https://mapzen-assets.s3.amazonaws.com/resources/tangram-work/tangram-work-darwin-x64.zip)

While you can run it yourself, we recommend that you only install it with the permission and assistance of your organization's IT staff. (Please note: we are not yet a registered developer on the Apple and Microsoft platforms. For Mac OS X, you may need to Control-click on the app to allow it to run. For Windows 10, if you see "Windows protected your PC", click "More info" then "Run anyway".)

As usual, we are open to your feedback and we thank you for trusting in Mapzen, your mapping solutions provider for over twenty years. Let us know what you think of the new Tangram Work!

· 01 April 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

The Who's On First API

whosonfirst (/tag/whosonfirst)

CGI
has a posse

(https://en.wikipedia.org/wiki/Common_Gateway_Interface)

Today we are pleased to announce **the Who's On First API (https://mapzen.com/documentation/wof/)**. This is something that has been hiding in plain sight for a little while now and that a few people may have noticed if they "looked under the hood" of the **Spelunker code (https://github.com/whosonfirst/whosonfirst-www-spelunker/)** following the recent blog post about the **Who's On First bundler tool (https://mapzen.com/blog/bundler/)**.

The API provides programmatic access for you and your robots to all the **Who's On First data (https://github.com/whosonfirst-data)**. You can query **individual places (https://mapzen.com/documentation/wof/methods/#whosonfirstplaces)** and their relations, look for **concordances (https://mapzen.com/documentation/wof/methods/#whosonfirstconcordances)** and perform **basic spatial queries (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getIntersects)**. You can also use the API to query things like all the different **placetypes (https://mapzen.com/documentation/wof/methods/#whosonfirstplacetypes)** or **sources (https://mapzen.com/documentation/wof/methods/#whosonfirstsources)** for data. It is not a "complete" API yet. The first goal for the API is to achieve parity with the **Spelunker (https://whosonfirst.mapzen.com/spelunker/)**: anything you can do by clicking around that website manually should be able to be automated using code.

You should treat this API as though it were still in "beta". Which is to say: the point is for *the thing to work* but there are probably still some rough edges and lingering gotchas so you should adjust your expectations and your code accordingly. In the meantime have at it and **please let us know on Twitter (https://twitter.com/alloftheplaces)** or **contact us through email (mailto:hello@mapzen.com)** if something is busted or just doesn't feel right.

If you want to dive in right away and come back for storytime later the documentation is here:

https://mapzen.com/documentation/wof/ (https://mapzen.com/documentation/wof/)

"Stuff over HTTP"

The API uses a parameter-based **stuff over HTTP (http://www.aaronland.info/weblog/2014/02/10/36hours/#what)** style interface. Currently all API methods are read-only so everything is sent using **the HTTP GET verb (https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3)**. If the method is successful the result with be returned with an HTTP `200 OK` response. If there was a problem with the request then an error will be returned in **the HTTP 400-499 range (https://mapzen.com/documentation/wof/errors/#client-side-errors)**. If there was a problem generating a response then an error will be returned in **the HTTP 500-599 range (https://mapzen.com/documentation/wof/errors/#server-side-errors)**.

Successful API responses can be returned in one of three possible formats: **JSON (https://mapzen.com/documentation/wof/formats/#json)**, **CSV (https://mapzen.com/documentation/wof/formats/#csv)** and Who's On First's own "**meta (https://mapzen.com/documentation/wof/formats/#meta)**" format (which is just a CSV file with fixed headers). JSON is the default response format and available for all methods. CSV is available for most methods. Meta responses are available for methods where it makes sense.

Meta files are the CSV files included with every **Who's On First data repository (https://github.com/whosonfirst-data)**. The easiest way to think about meta files is that they canned indices or views in to the data; a way for people to grab or work with a slice of the data, say by placetype, without having to set up and load everything in to a database. As such, a lot of tooling has been built to consume meta files from the **I Am Here (https://mapzen.com/blog/iamhere)** tool, to the Who's On First **point-in-polygon server (https://github.com/whosonfirst/go-whosonfirst-pip/)**, to the tools we use to **generate bundles (https://github.com/whosonfirst/go-whosonfirst-clone#example)** (you can read more about bundles **over here (https://whosonfirst.mapzen.com/bundles/)**), to indexing WOF data in **in third-party tools like Tile38 (https://github.com/whosonfirst/go-whosonfirst-**

tile38#example-1) to **generating FeatureCollections (https://github.com/whosonfirst/py-mapzen-whosonfirst-utils/blob/master/scripts/wof-csv-to-feature-collection)**. Now rather than just consuming the **default meta files (https://github.com/whosonfirst-data/whosonfirst-data/tree/master/meta)** included with each repository all of those tools can be fed the output of whatever query you dream up using the **whosonfirst.places.search (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.search)**, or similar, API methods.

(https://collection.cooperhewitt.org/objects/18214745/)

Other API responses can and will be added as time and circumstance permit. We are interested in people *using* the Who's On First API in whatever format their software needs and with not in making our opinions about API responses someone else's problem. If there's a particular response format you need to start using the Who's On First API please send up a flare **via email (mailto:hello@mapzen.com)** or **Twitter (https://twitter.com/alloftheplaces)**.

Examples

whosonfirst.concordances.getById (https://mapzen.com/documentation/wof/methods/#whosonfirst.concordances.getById)

Here are all the concordances in Who’s On First for the **GeoPlanet (https://whosonfirst.mapzen.com/spelunker/concordances/geoplanet/)** ID for the city of **Montreal (https://whosonfirst.mapzen.com/spelunker/101736545)** or `gp:id=3534` :

```
curl -s -X GET 'https://whosonfirst-api.mapzen.com?method=whosonfirst.concordances.getById
{
    "concordances": [
        {
            "dbp:id": "Montreal",
            "fb:id": "en.montreal",
            "fct:id": "03c06bce-8f76-11e1-848f-cfd5bf3ef515",
            "gn:id": 6077243,
            "gp:id": 3534,
            "loc:id": "n80132975",
            "nyt:id": "N59179828586486930801",
            "qs:id": "239659",
            "tgn:id": "7013051",
            "wd:id": "Q340",
            "wk:page": "Montreal",
            "wof:id": 101736545
        }
    ],
    "cursor": null,
    "next_query": null,
    "page": 1,
    "pages": 1,
    "per_page": 100,
    "stat": "ok",
    "total": 1
}
```

whosonfirst.places.getByLatLon (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getByLatLon)

Here are all the neighbourhoods (there’s only one of them) at the corner of **16th and Mission (https://whosonfirst.mapzen.com/iamhere/#17/37.76494/-122.41944)** in San Francisco:

```
curl -s -X GET 'https://whosonfirst-api.mapzen.com?method=whosonfirst.places.getByLatLon&a
{
    "places": [
        {
            "geom:latitude": 37.758768,
            "geom:longitude": -122.413313,
            "mz:uri": "https://whosonfirst.mapzen.com/data/858/874/43/85887443.geojson",
            "wof:country": "US",
            "wof:id": 85887443,
            "wof:name": "Mission District",
            "wof:parent_id": "85922583",
            "wof:placetype": "neighbourhood",
            "wof:repo": "whosonfirst-data"
        }
    ],
    "stat": "ok"
}
```

There are a couple things to note about this API response. First, the "**minimum response data (http://code.flickr.net/2008/08/19/standard-photos-response-apis-for-civilized-age/)**" for a place and second the use of the `extras` parameter to include additional information about a place. We're still working out what the final "minimum response" data structure for a place should be but so far we've settled on the following properties:

```
{
    "wof:id": ...,
    "wof:name": "...",
    "wof:parent_id": ...,
    "wof:placetype": "...",
    "wof:repo": "..."
}
```

Every API method that returns a "place" as part of its response will be guaranteed to include those properties. If you need or want additional properties you can list them in the `extras` parameter. Passing the `extras=geom:latitude,geom:longitude` parameter will cause the API to add the geographic center a place's geometry to its response. You can also request entire classes of properties by passing only a prefix.

whosonfirst.places.getInfo (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getInfo)

For example to fetch **all the names (https://mapzen.com/blog/wikipedia-data/)** for the city of Beijing you would pass `extras=name:` , like this:

```
curl -s -X GET 'https://whosonfirst-api.mapzen.com?method=whosonfirst.places.getInfo&api_k
{
    "place": {
        "name:ace_x_preferred": [
            "Beijing"
        ],
        "name:ady_x_preferred": [
            "\u041f\u0435\u043a\u0438\u043d"
        ],
        "name:afr_x_preferred": [
            "Beijing"
        ],
        "name:als_x_preferred": [
            "Peking"
        ],
        "name:amh_x_preferred": [
            "\u1264\u12ea\u1302\u1295\u130d"
        ],
        "name:ang_x_preferred": [
            "Beicing"
        ],

    ...truncated for brevity...

    "name:zho_x_preferred": [
            "\u5317\u4eac\u5e02"
        ],
        "name:zho_x_variant": [
            "\u5317\u4eac\u5e02"
        ],
        "name:zho_yue_x_preferred": [
            "\u5317\u4eac"
        ],
        "wof:country": "CN",
        "wof:id": 102027745,
        "wof:name": "Beijing",
        "wof:parent_id": "85669727",
        "wof:placetype": "locality",
        "wof:repo": "whosonfirst-data"
    },
    "stat": "ok"
}
```

You can request any fully-qualified property, or property prefix, in the `extras` parameter. The API will check to ensure that the prefix for an extras parameter is defined in the **whosonfirst-sources (https://github.com/whosonfirst/whosonfirst-sources/tree/master/sources)** list (there are **corresponding API methods for sources (https://mapzen.com/documentation/wof/methods/#whosonfirstsources)**). As of this writing if an unknown prefix is requested it is silently ignored. *That might become an error in time. We'll see…*

whosonfirst.places.search (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.search)

Here are all the **microhoods (https://whosonfirst.mapzen.com/spelunker/placetypes/microhood)** in Who's On First returned as a CSV document:

```
curl -s -i -X GET 'https://whosonfirst-api.mapzen.com?method=whosonfirst.places.search&api
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: text/csv
Date: Fri, 31 Mar 2017 21:41:50 GMT
Server: nginx/1.4.6 (Ubuntu)
Status: 200 OK
X-api-format-csv-header: wof_country,wof_id,wof_name,wof_parent_id,wof_placetype,wof_repo
X-api-pagination-cursor: cXV...c7MDs=
X-api-pagination-next-query: method=whosonfirst.places.search&amp;placetype=microhood&amp;
X-api-pagination-page:
X-api-pagination-pages: 60
X-api-pagination-per-page: 10
X-api-pagination-total: 595
Content-Length: 746
Connection: keep-alive

wof_country,wof_id,wof_name,wof_parent_id,wof_placetype,wof_repo
US,1108802091,"Scripps Miramar Ranch",1108802089,microhood,whosonfirst-data
US,1108561153,"La Tuna Canyon",85865489,microhood,whosonfirst-data
US,1108750019,"Green Valley Ranch",420781629,microhood,whosonfirst-data
US,1075806299,"Northridge West",85838305,microhood,whosonfirst-data
DE,1108810201,Biesdorf-Sud,-3,microhood,whosonfirst-data
US,1108719769,"Rivergate Industrial District",85846673,microhood,whosonfirst-data
US,1108750017,Gateway,420781629,microhood,whosonfirst-data
US,1108750051,"Northeast Park Hill",85840547,microhood,whosonfirst-data
US,1091648325,"Hansen Dam",85865477,microhood,whosonfirst-data
US,1041491317,"Beverly Glen",85869119,microhood,whosonfirst-data
```

There are a couple of things to note about CSV formatted responses:

- The `:` character that normally separates a prefix from its value has been replaced by a `_`. For example `wof:id` becomes `wof_id` when it is encoded as a CSV column header. It's not awesome but lots of tools that consume CSV have problems with column headers containing colons so we just live with the inconsistency.
- Nested values (arrays and dictionaries) are returned as JSON-encoded strings. It is left up to consumers of the API to decode them as necessary.

Also, did you notice the many different pagination properties in the JSON responses and the HTTP headers for the CSV response? Yeah…

A short miserable history of pagination

(https://collection.cooperhewitt.org/objects/18805295/)

Pagination, the practice of chunking a lot of results in to smaller sets, shouldn't be complicated. But it is. Because databases, after all these years, are still complicated beasts.

Databases have always been about trade-offs. No two databases are the same and so no two sets of trade-offs are the same either. The really short version is that some databases can't tell you exactly how many results there are for a given query. Some databases can tell you how many results there are but can't or won't return results past a certain limit. Other databases can do both but only if you use something called a cursor for pagination rather than the traditional offset and limit model (as in "return the next 5 of 50 results starting from postion 20").

A long time ago, **I worked at the photo-sharing website Flickr (https://www.flickr.com/photos/paulhammond/2883048077/)**. One of the limits of the **Flickr API (https://www.flickr.com/services/api)** is that the `flickr.photos.search` method was, and still is, capped at 4,000 results for any single query. This was a limit of **Flickr's search engine (http://aaronland.info/talks/mw10_machinetags/#92)**, an early and proprietary version of applications like **Solr (https://lucene.apache.org/solr/)** and more recently **Elasticsearch (https://www.elastic.co/guide/en/elasticsearch)** which are often referred to as "document stores".

Document stores are similar to relational databases (think of the many different `something-SQL` databases people talk about) in that you can query them and get stuff back. Relational databases and document stores are very different in how they accomplish the same task and one of the limitations of the latter has always been that they aren't really designed to return the "long tail" of results for a query with lots of results.

When I last checked the Flickr website it returns **1,844,154 photos when you search for kittens (https://www.flickr.com/search/?text=kittens)**. If you query the Flickr API for photos tagged "kittens" it says there are 429,203 photos. The disparity is because people are allowed to opt-out of including their photos in API results.

Here is the first result at around the 4,000 mark (500 photos per request * page 8, out of a possible 859):

```
https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=*****&tags=kitte

{ "photos": { "page": 8, "pages": "859", "perpage": "500", "total": "429203",
    "photo": [
          { "id": "33746803785", "owner": "148036032@N07", "secret": "4f91e6622b", "server
```

And here's the same query but at the 5,000 mark (500 photos per request * page 10):

```
https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=*****&tags=kitte

{ "photos": { "page": 10, "pages": "859", "perpage": "500", "total": "429203",
    "photo": [
          { "id": "33746803785", "owner": "148036032@N07", "secret": "4f91e6622b", "server
```

See what's happening there? It's the same photo at the beginning of the list even though the results are a thousand photos apart. The search engine is configured to return the same results for every request over the 4,000 results limit. This is not a new thing. This is also not a Flickr thing. It's a document store thing. Even today, ten years, later the state of the art hasn't improved all that much: In 2017, Elasticsearch still has **a hard limit of 10,000 results (https://www.elastic.co/guide/en/elasticsearch/guide/current/pagination.html)** for a single query.

The reason I mention all of this is not to pick on Flickr or Elasticsearch or anyone else working with document stores. The performance of "traditional" relational databases is also known to increasingly degrade as you offset further and further in to a large result set (the rule of thumb is that things start to get painful around the 500,000 record mark). The limits imposed by document stores are often worth it because they allow you perform complex queries that would otherwise be impossible or impractical in another database. I mention all of this for two reasons:

First, there remains a lot of interesting and important work to do designing interfaces and models to account for the inability of databases to reliably (or efficiently) return all the results for a large query. The 4,000 results limit was **well-known by developers (https://www.flickr.com/groups/api/discuss/72157663968579450/)** using the Flickr API. We didn't hide that it happened but we also didn't see it as an opportunity to encourage developers to think about different ways to imagine what search should look like or how people should think about using it. The work that **George Oates (https://en.wikipedia.org/wiki/George_Oates)** did redesigning the search and subject pages for the **Internet Archive's Open Library project (https://archive.org/details/Kohacon2010OpenLibraryPresentation-GeorgeOates)** is a good example of how we might approach that problem, going forward.

Second, the reason for telling you all of this "exciting" detail from the past is to help explain how and why we've implemented pagination for the Who's On First API in the present. Who's On First is all about **the relationship between places (https://whosonfirst.mapzen.com/placetypes)** and sometimes a place (like a country) will have lots and lots of relationships (like all the venues in that country). We don't really have the luxury of "ranking" those relationships and only returning the top 4,000 (or 10,000) results. We need to find a way to return **all the things (https://mapzen.com/blog/all-of-the-places)**.

Further, Who's On First uses multiple, different databases by design. We do this to ensure that the data and the overall data modeling is flexible enough to work with as many tools as possible. That's very important to the project because we want to make sure that the infrastructure burden required to *do* something with Who's On First data is as a light as possible and not a reflection of Mapzen's specific needs to operationalize things or the inertia of our own preferences.

For example the **Spelunker (https://whosonfirst.mapzen.com/spelunker)** is built using Elasticsearch but has no spatial indices even though they are supported. We set things up this way to make sure, to prove, that it was possible to use the data without *requiring* a spatial database.

Since there is no all-purpose database, the Who's On First API accounts for multiple different pagination models. We've identified four overlapping models (**plain**, **cursor**, **mixed** and **next-query**) each of which are described in detail below. If you don't really care and just want to get started **you should skip ahead to the discussion of next-query pagination**.

Plain pagination

Plain pagination assumes that we know how many results a query yields and that we can fetch any set of results at any given offset. For example, let's say you wanted to use the API to fetch all the places with a variant name containing the word `Paris` in sets of five. The API will respond with something like this:

```
{
    "places": [ ... ],
    "next_query": "method=whosonfirst.places.search&alt=Paris&per_page=5&page=2",
    "total": 7,
    "page": 1,
    "per_page": 5,
    "pages": 2,
    "cursor": null,
    "stat": "ok"
}
```

It's pretty straightforward. There are seven results (`total`) and this is the first of two pages worth of results (`page` and `pages` , respectively). You might already be wondering about the `next_query` property but **we'll get to that shortly**.

Cursor-based pagination

Cursor-based pagination is necessary when a database can't or won't tell you how many results there are for a query. This means you will need to pass the same query to the database over and over again for as long as the database returns a `cursor` which is like a secret hint *that only the database understands* indicating where the next set of results live.

For example, let's say you wanted to use the API to fetch all of the venues near the **Smithsonian Cooper Hewitt Design Museum (https://whosonfirst.mapzen.com/spelunker/id/420571601/)** in sets of ten. The API will respond with something like this:

```
{
    "places": [ ... ],
    "next_query": "method=whosonfirst.places.getNearby&latitude=40.784165&longitude=-73.95
    "per_page": 10,
    "cursor": {CURSOR},
    "stat": "ok"
}
```

In order to fetch the next set of results you would include a `cursor={CURSOR}` parameter in your request, rather than a `page={PAGE_NUMBER}` parameter like you would with plain pagination. Some databases yield time-sensitive cursors that expire after a number of seconds or minutes so the easiest way to think about cursors is that they are *all* time sensitive.

Databases, amirite?

Mixed pagination

This is where it gets fun. Sometimes an API method might use *both* plain and cursor-based pagination. That can happen when an underlying database is able to calculate the total number of results but only be able to fetch a fraction of them using plain pagination after which it needs to switch to cursor-based pagination. Which doesn't really make any sense when you think about it because cursors are magic database pixie-dust so there's no way to determine or calculate a corresponding cursor for a traditional page number. So in the end the API itself needs to perform an initial query just to see how many results there are and then adjust whether it is going to use plain or cursor-based pagination on the fly.

For example, let's say you wanted to use the API to fetch all the `microhoods` in sets of five. The API will respond with something like this:

```
{
    "places": [ ... ],
    "next_query": "method=whosonfirst.places.search&placetype=microhood&page=2&per_page=5"
    "total": 186,
    "page": 1,
    "per_page": 5,
    "pages": 38,
    "cursor": null,
    "stat": "ok"
}
```

But if you then asked the API to fetch all of the `neighbourhoods`, again in sets of five, the API will respond with something like this:

```
{
    "places": [ ... ],
    "next_query": "method=whosonfirst.places.search&placetype=neighbourhood&per_page=5&cur
    "total": 81065,
    "page": null,
    "pages": 16213,
    "per_page": 5,
    "cursor": "{CURSOR}",
    "stat": "ok"
}
```

In both examples we know how many results there will be. In the first example we are able to use plain pagination so we know that this is page one of thirty-eight and thus the value of the `cursor` property is null. In the second example the API has returned a cursor so even though we know the total number of results and can calculate the number of “pages” we set the value of the `page` property to be null since the requirement on cursor-based pagination makes it moot.

If you look carefully at the value of the `next_query` property in both examples you can probably figure out where this is going, next.

Next-query-based pagination

Next-query based pagination is an attempt to hide most of the implentation details from API consumers and provide a simple “here-do-this-next” style pagination interface, instead.

For example, let’s say you wanted to use the API to fetch all the localities (there are over 200,000 of them) in sets of five. That will require more than 41,000 API requests but that’s your business. The API will respond with a `next_query` parameter, something like this:

```
{
    "places": [ ... ],
    "next_query": "method=whosonfirst.places.search&placetype=locality&per_page=5&cursor={
    "total": 208214,
    "page": null,
    "pages": 41643,
    "per_page": 5,
    "cursor": "{CURSOR}",
    "stat": "ok"
}
```

There are a few things to note about the `next_query` property:

- It contains a URL-encoded query string with the parameters to pass to the API retrieve the *next* set of results for your query.
- When it is empty (or `null`) that means there are no more results.
- It *does not* contain any user-specific access tokens or API keys — you will need to add those yourself.
- It *does not* contain any host or endpoint specific information — you will need to add that yourself.
- You may want or need to decode the query string in order to append additional parameters (like authentication) and to handle how those parameters are sent along to the API. For example, whether the method is invoked using HTTP’s `GET` or `POST` method or whether parameters should be `multipart/mime` encoded or not. And so on.

This type of pagination is not ideal but strives to be a reasonable middle-ground that is not too onerous to implement and easy to use.

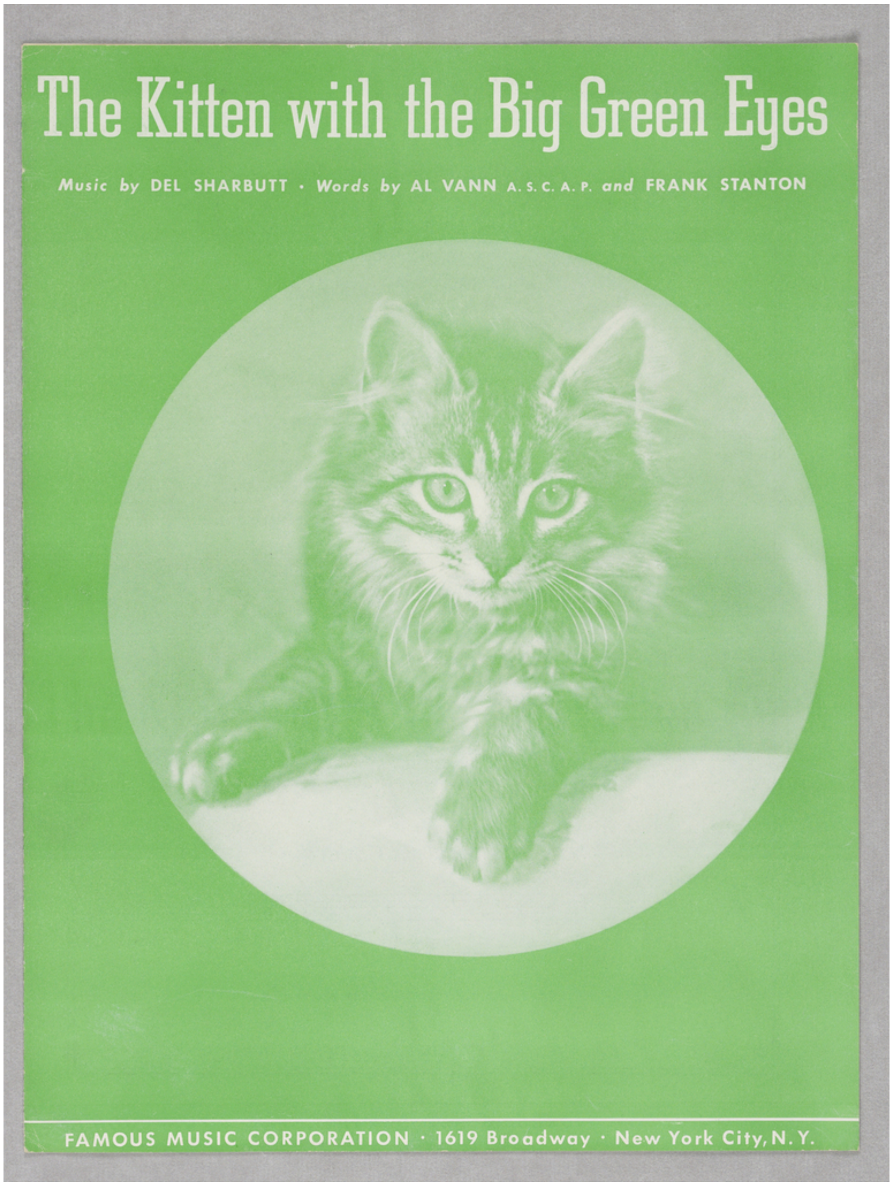
Pagination and HTTP headers

Pagination properties are also returned as HTTP response headers. This is useful for any output format and necessary for output formats like plain old **CSV** or Who’s On First’s **meta** format. All of the pagination properties you’ve come to know and love in the examples above are also returned as HTTP response header prefixed by `X-api-pagination-` .

For example:

```
$> curl -s -v -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.search

< HTTP/1.1 200 OK
< Access-Control-Allow-Origin: *
< Content-Type: text/csv
< Date: Tue, 28 Feb 2017 21:13:37 GMT
< Status: 200 OK
< X-api-pagination-cursor:
< X-api-pagination-next-query: method=whosonfirst.places.search&amp;q=poutine&amp;extras=g
< X-api-pagination-page: 1
< X-api-pagination-pages: 13
< X-api-pagination-per-page: 1
< X-api-pagination-total: 13
< X-whosonfirst-csv-header: geom_bbox,wof_country,wof_id,wof_name,wof_parent_id,wof_placet
< Content-Length: 208
< Connection: keep-alive
<
geom_bbox,wof_country,wof_id,wof_name,wof_parent_id,wof_placetype,wof_repo
"-71.9399642944,46.0665283203,-71.9399642944,46.0665283203",CA,975139507,"Poutine Restau-B
```

Libraries


The Kitten with the Big Green Eyes
Music by DEL SHARBUTT · Words by AL VANN A.S.C.A.P. and FRANK STANTON
FAMOUS MUSIC CORPORATION · 1619 Broadway · New York City, N.Y.

(https://collection.cooperhewitt.org/objects/18612725/)

As of this writing there are two general-purpose software libraries for accessing the Who's On First API, one written in **Python (https://github.com/whosonfirst/py-mapzen-whosonfirst-api)** and another in **Go (https://github.com/whosonfirst/go-whosonfirst-api)**. We also have PHP and Javascript libraries that we use internally but they are still full of Mapzen-isms which we'd like to clean up before releasing.

The Python and Go libraries don't have feature parity and probably never will. They are each well-suited for different tasks and, as with databases, we use their respective strengths and weaknesses to test the decisions we make modeling the actual Who's On First data. We use both of these libraries on a day-to-day basis. *Most* of the kinks should be worked out by now but the documentation could be improved in places, notably in the **Go (https://github.com/whosonfirst/go-whosonfirst-api)** package which we're using **as a way to test (https://github.com/whosonfirst/go-whosonfirst-api#interfaces)** how a strictly typed language can work with less formal and semi-structured API responses.

If you write your own library for the Who’s On First API **we’d love to hear about it (https://twitter.com/alloftheplaces)**.

py-mapzen-whosonfirst-api

The Python library is available on GitHub: **https://github.com/whosonfirst/py-mapzen-whosonfirst-api (https://github.com/whosonfirst/py-mapzen-whosonfirst-api)**. Here’s a simple example of how you might use it:

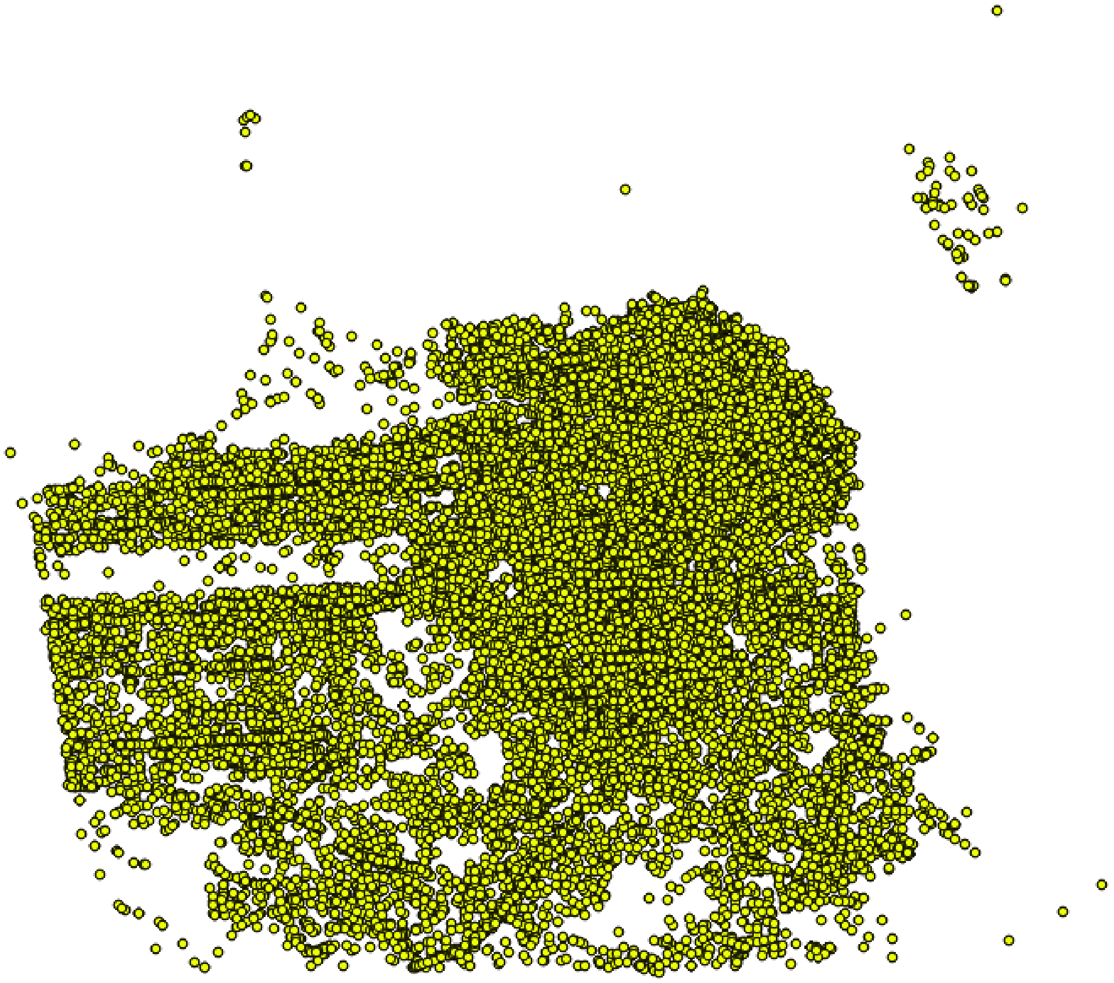
```
import mapzen.whosonfirst.api.client

api = mapzen.whosonfirst.api.client.Mapzen("mapzen-xxxxxx")
print api.execute_method("api.spec.formats", {})

# prints:
# {u'default_format': u'json', u'stat': u'ok', u'formats': [u'json', u'csv', u'meta']}
```

go-whosonfirst-api

The Go library is available on GitHub: **https://github.com/whosonfirst/go-whosonfirst-api (https://github.com/whosonfirst/go-whosonfirst-api)**. Here’s a simple example of how you might use it:

```
import (
    "github.com/whosonfirst/go-whosonfirst-api/client"
    "github.com/whosonfirst/go-whosonfirst-api/endpoint"
    "os"
    )

func main() {

    api_key := "mapzen-xxxxxx"

    api_endpoint, _ := endpoint.NewMapzenAPIEndpoint(api_key)
    api_client, _ := client.NewHTTPClient(api_endpoint)

    method := "whosonfirst.places.search"

    args := api_client.DefaultArgs()
    args.Set("query", "poutine")
    args.Set("placetype", "venue")

    rsp, _ := api_client.ExecuteMethod(method, args)
    os.Stdout.Write(rsp.Raw())

    # prints (truncated for brevity)
    # {"places":[{"wof:id":152777717,"wof:parent_id":"85874363","wof:name":"Poutine LA Fle
}
```

There is also a command line tool called `wof-api` for performing simple and batch operations with the API. For example, using the tool it’s possible to **fetch all 63,387 venues in San Francisco (https://github.com/whosonfirst/go-whosonfirst-api#example)** as a single GeoJSON FeatureCollection. There are lots of ways to do this but the nice thing about the `wof-api` tool is that depending on your network connection this can be accomplished in as little as 5 minutes.



All those dots in the image above were produced by typing `wof-api -param method=whosonfirst.places.search -param locality_id=85922583 -param api_key=mapzen-xxxxxx -param per_page=500 -param placetype=venue -paginated -geojson -output venues.geojson -timings -async`

Note the use of the `-paginated` flag which will instruct the tool to handle all the pagination nonsense described above for you. It’s kind of like having your very own **Bundler (https://mapzen.com/blog/bundler/)** on the **command line (https://en.wikipedia.org/wiki/In_the_Beginning..._Was_the_Command_Line)**.

Inspirational conclusion

HAND SHADOWS.
THE CAMEL.
THE GOOSE.
THE DOG.
THE GOAT.
ICH DIEN
ICH DIEN
BUNNY.

(https://collection.cooperhewitt.org/objects/18402495/)

We hope you make awesome things with the API. Enjoy!

Image credits:

- **Model Of A Walled Entrance With Gate (Italy), ca. 1710; carved and painted pine, wrought iron gate and window grill; Gift of Mrs. James O. Green and Eleanor and Sarah Hewitt; 1921-17-16-a/d (https://collection.cooperhewitt.org/objects/18214745/)**
- **Label (USA), 1980–85; jacquard woven; 1985-39-60 (https://collection.cooperhewitt.org/objects/18805295/)**
- **Sheet Music, The Kitten with the Big Green Eyes; 1949-106-3 (https://collection.cooperhewitt.org/objects/18612725/)**
- **Panel (England), mid-19th century; cotton; Gift of Ruth Van Norman; 1955-119-1 (https://collection.cooperhewitt.org/objects/18402495/)**

· 04 April 2017 ·

Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

Statistical NLP on OpenStreetMap, Part 2

search (/tag/search)

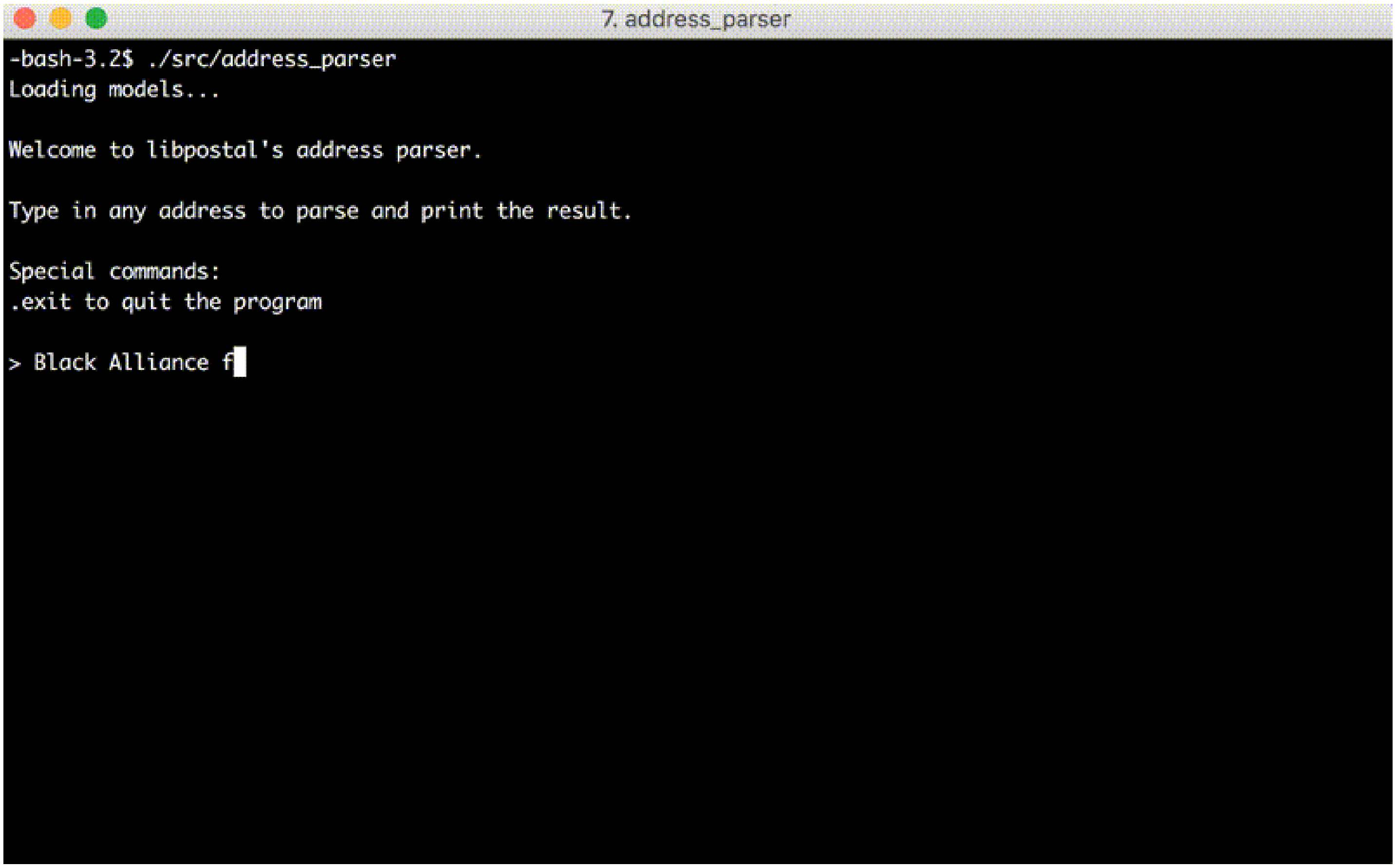
Data scientist Al Barrentine has been working with Mapzen to crack one of the hardest problems in geocoding and place search: international address parsing. He continues to improve libpostal, a state-of-the-art, lightning-fast C library and statistical model for parsing and normalizing addresses around the world. We use it in Mapzen Search, and are happy to see that over the past year the parser has increased to 99.45% full-parse accuracy! The full post is ***on Medium (https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-part-2-80405b988718)****.*

A while back I wrote **a piece (https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-b9d573e6cc86)** publicly introducing **libpostal (https://github.com/openvenues/libpostal)**, an open-source, open-data-trained C library and companion NLP model for parsing and normalizing international street addresses. Since then, libpostal's user base has grown to include governments, startups, large companies, researchers, and data journalists from over a dozen countries around the world.

Today I'm very excited to announce the release of libpostal 1.0, featuring a new address parser trained on over 1 billion examples in every inhabited country on Earth from great open data sets like **OpenStreetMap (https://openstreetmap.org)** and **OpenAddresses (https://openaddresses.io)**. There are several new tags for complex sub-building structures (unit, floor, staircase, etc.), P.O. boxes, and categorical queries, each handling around 30 different languages. Best of all, the address parser now features a much more powerful machine learning model (a Conditional Random Field or CRF), which can infer a globally optimal tag sequence instead of making local decisions at each word.

The new model achieves 99.45% full-parse accuracy on held-out addresses (i.e. addresses from the training set that were purposefully removed so we could evaluate the parser on addresses it hasn't seen before). This is more significant than just a half a percent improvement over the original parser's 98.9% result because now it has to deal with more labels and more languages/countries.

The new parser in GIF form



```
-bash-3.2$ ./src/address_parser
Loading models...

Welcome to libpostal's address parser.

Type in any address to parse and print the result.

Special commands:
.exit to quit the program

> Black Alliance f
```

You can ***read the full post on Medium (https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-part-2-80405b988718)****, but here's a quick list of the topics Al covers:*

East Asian system addresses, randomizing address formats, admin boundary mappings, no umlaut left behind, reverse geocoding to buildings, point-based places and neighborhoods, a places-only data set, a streets-only data set, OpenAddresses, GeoPlanet postal codes, partments/sub-building components, units, level & floors, staircases, entrances, PO boxes, simple place name queries, consistency, abbreviated toponyms, category queries, chain stores, feature extraction, N-grams for rare/unknown words, hyphenated sub-words, right-context phrases, postcode contexts, Conditional Random Fields FTW, exact inference, greedy parsing (YOLO method), CRF parsing (Spock method), a fast, scalable CRF implementation, state-transition features, sparser perceptrons, 100GB of public training data.

· 07 April 2017 ·

Al Barrentine

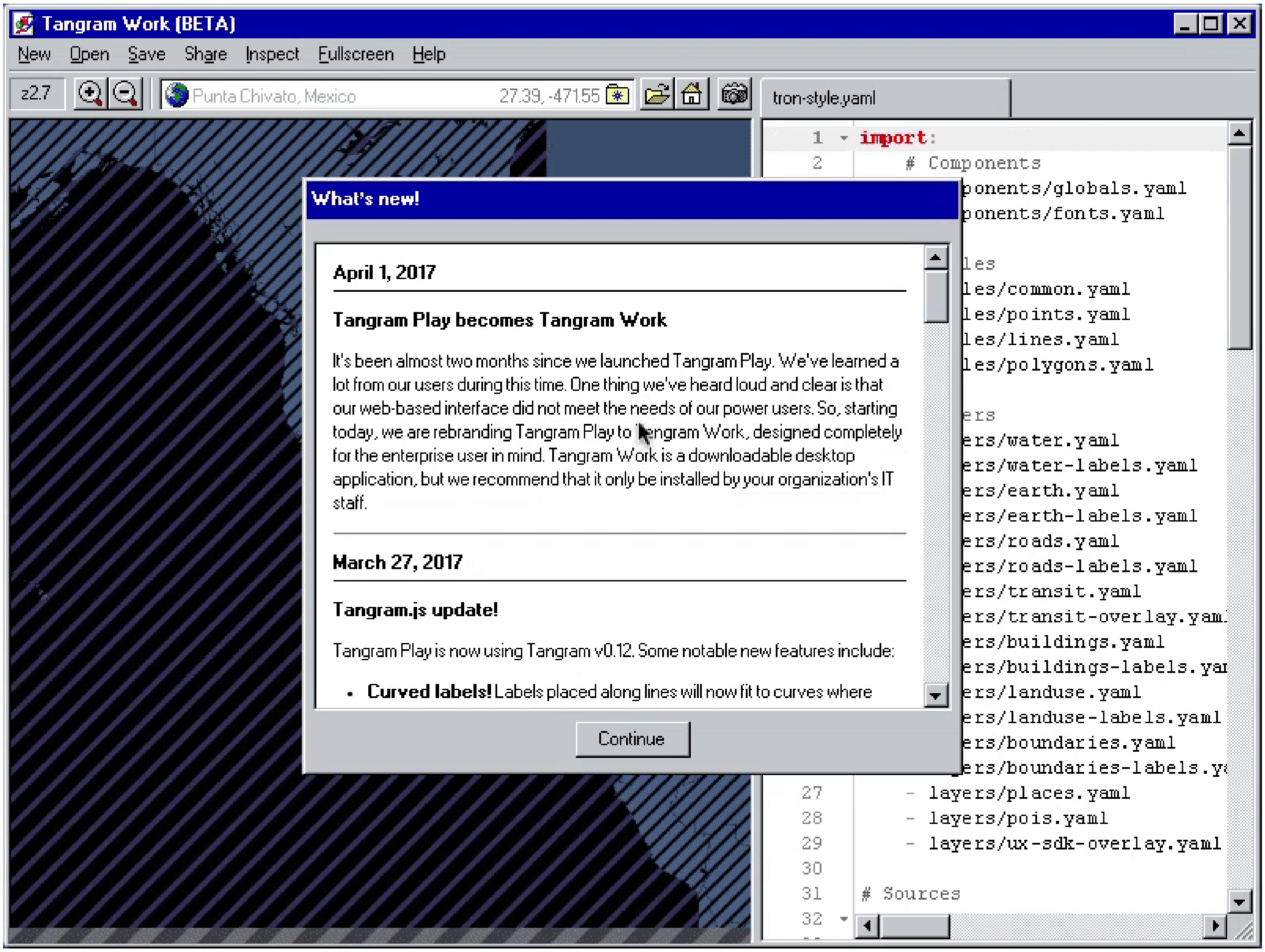
Al Barrentine is a Brooklyn-based data scientist passionate about open data, and believes high-quality data sets like Wikidata, OpenStreetMap, and Common Crawl are essential to creating a culture of reproducibility in machine learning.

© 2017 Mapzen

Tangram Work: An April Fool's Day developer commentary

tangram (/tag/tangram)

April Fool's Day isn't for everyone, so if you were one of the people who spent that weekend hiding from the Internet and other forms of media, you might have missed our first-ever April Fool's joke. (We're sorry. Also, not sorry.) We showed off a "pivot" of **Tangram Play (https://mapzen.com/tangram/play)**, reskinned as a Windows 95 application and **rebranded as Tangram Work (https://mapzen.com/blog/introducing-tangram-work/)**, targeted for "power users in an environment environment." It was a fully functional downloadable application, carefully crafted to be as close to pixel perfect as we could make it to evoke an older era of personal computing.



April 1, 2017

Tangram Play becomes Tangram Work

It's been almost two months since we launched Tangram Play. We've learned a lot from our users during this time. One thing we've heard loud and clear is that our web-based interface did not meet the needs of our power users. So, starting today, we are rebranding Tangram Play to Tangram Work, designed completely for the enterprise user in mind. Tangram Work is a downloadable desktop application, but we recommend that it only be installed by your organization's IT staff.

March 27, 2017

Tangram.js update!

Tangram Play is now using Tangram v0.12. Some notable new features include:

- **Curved labels!** Labels placed along lines will now fit to curves where

```
import:
    # Components
ponents/globals.yaml
ponents/fonts.yaml

les
les/common.yaml
les/points.yaml
les/lines.yaml
les/polygons.yaml

ers
ers/water.yaml
ers/water-labels.yaml
ers/earth.yaml
ers/earth-labels.yaml
ers/roads.yaml
ers/roads-labels.yaml
ers/transit.yaml
ers/transit-overlay.yam
ers/buildings.yaml
ers/buildings-labels.yar
ers/landuse.yaml
ers/landuse-labels.yaml
ers/boundaries.yaml
ers/boundaries-labels.ya
    - layers/places.yaml
    - layers/pois.yaml
    - layers/ux-sdk-overlay.yaml

# Sources
```

Nostalgia is a rich vein of material from which to **mine (https://minecraft.net/en-us/article/introducing-mine-craft-digital-leisure-device)** April Fool's jokes, and good parodies are exaggerations of real experiences. (Many of us have probably sat down at a computer with an ancient version of office productivity software at some point.) But lest we ruin the joke by analyzing it too closely, let's talk about how we pulled this off and what we've learned.

A brief history of the World Wide Web

At some point in one's life, one should take a moment to realize how utterly absurd it is that the thing we call "the web," despite its flaws and its organically patched-up architecture, came to take over **all of computing (https://www.destroyallsoftware.com/talks/the-birth-and-death-of-javascript)**. The initial version of JavaScript, the now-ubiquitous programming language of the web, was **created in ten days (https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript)** back in 1995. It only worked in **one browser (https://en.wikipedia.org/wiki/Netscape_(web_browser))**. When it was adopted by other browser vendors, incompatible implementations kept it in the realm of a plaything for amateurs. Professional programmers, raised speaking languages like C, despised it for its illogical quirks.

And then, over time, it got better. Today, JavaScript rules everything around us.

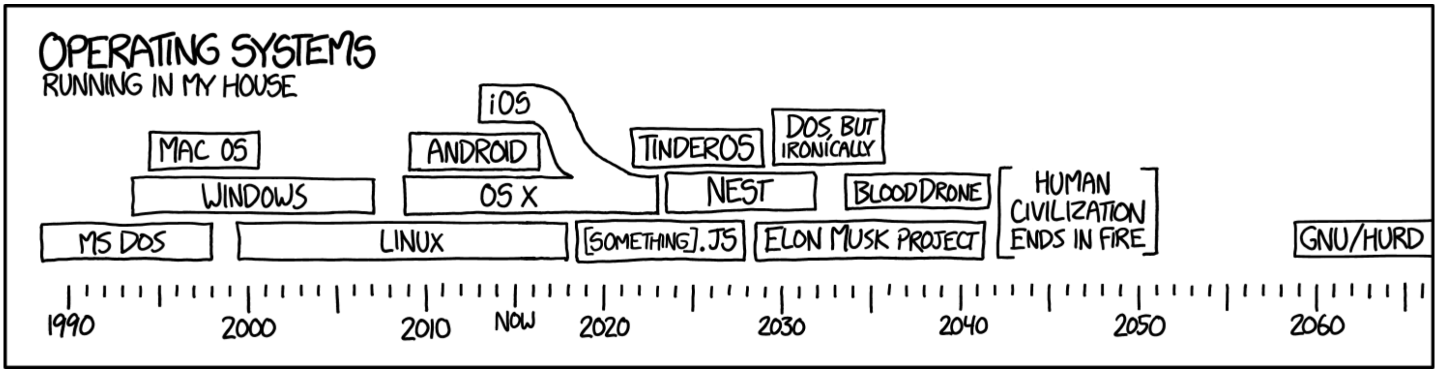
Depending on your mood, **server-side JavaScript (https://nodejs.org/)** is either the earliest criminal offender or a celebrated pioneer of legend. With Node.js, front-end developers could write server code in the same language they used on the client. That, too, had rough beginnings, but when the back-end lands were tamed and settled (or, if you prefer, invaded and occupied), developers set their sights on further uses (or abuses) of JavaScript by packaging the entire full stack of web infrastructure into something you could download from the App Store.

In 2013, GitHub released the **Atom (https://atom.io/)** text editor, which adopted both Node.js and **Chromium (https://www.chromium.org/Home)**, the open-source core of Google's Chrome browser, so that they could build a hackable text editor out of eminently hack-friendly web technologies. Its engineers realized that they had a reusable general-purpose framework on their hands, so they released it as a standalone project called **Electron (https://electron.atom.io/)**.

Other groups started to take advantage of this powerful tool. A small team can quickly prototype a product on Mac, Windows, and Linux simultaneously, and still offer a web-based version from the same codebase if they chose. But a project can also be incredibly efficient with no web presence at all: since the app bundled its own Chromium, there was no more need to worry about browser versions or cross-browser compatibility ever again. Like Java, it promised "write once, run anywhere."

This flexibility has led to an explosion of desktop applications made completely out of HTML, CSS, and JavaScript. An app you use every day, perhaps even one you have open right now, might be built on Electron. You may be using Slack in your office or online community — **that's Electron**

(https://slack.engineering/building-hybrid-applications-with-electron-dc67686de5fb). Or perhaps you write e-mail in Nylas, or program in Visual Studio Code, or chat in Discord… **the list goes on (https://electron.atom.io/apps/)**. I personally use the **Hyper (http://hyper.is/)** terminal to give me a command line interface — that powerful and most venerable method of telling a computer what to do — a 1960s-era innovation that in 2017, ironically, *requires an entire web browser to run*.



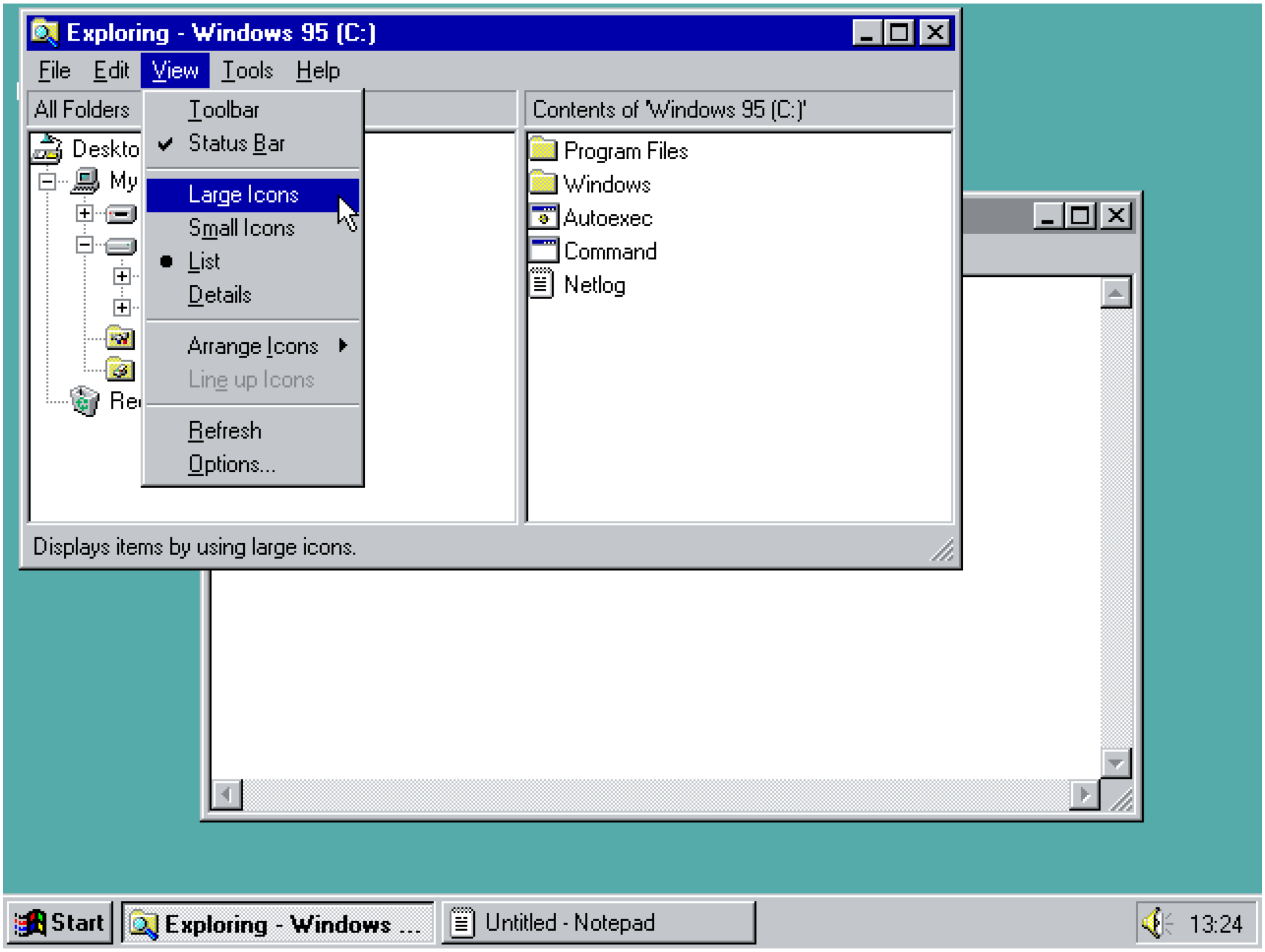
In 2015, xkcd predicted that in a couple of years, there will be a mainstream operating system based on JavaScript.

Electron became very appealing to experiment with, for a number of reasons. First, we wanted to fully sell the nostalgia by making Tangram Work an actual downloadable application that you ran by double-clicking an icon. But there was a pragmatic reason as well. We're planning to support reading and writing of many files related to a scene, including imported scenes, textures, sprites, data sources, and ZIP bundles. The hard way to do this is to invent a virtual file system that lives inside your browser memory. An alternative way is to give Tangram Play access to your actual file system in the first place. This isn't possible for a normal web browser, since browser vendors spend a lot of effort to make sure nothing malicious on the web can break out of its sandbox and read or write arbitrary files directly to your system. But a native application can do this without hassle.

And, in order to accurately recreate the Windows 95 experience in a short amount of time, it would be much, much easier if we only needed to work in one version of browser.

Only 90's kids will remember this

Despite Microsoft's near-constant tweaking of the Windows interface in each of its releases over the last two decades, the vanilla Windows 95 interface seems to be the most iconic and recognizable one. Initially, we'd assumed that other people might have already reproduced its look and feel, and that meant we might be able to poach some stylesheets from prior art. But a cursory search turned up precious little besides a handful of attempts, which didn't contain enough detail to work with. And then we discovered emulators.


Exploring - Windows 95 (C:)
File Edit View Tools Help
All Folders
Contents of 'Windows 95 (C:)'
Toolbar
Status Bar
Large Icons
Small Icons
List
Details
Arrange Icons
Line up Icons
Refresh
Options...
Program Files
Windows
Autoexec
Command
Netlog
Start
Exploring - Windows ...
Untitled - Notepad
13:24

Displays items by using large icons.

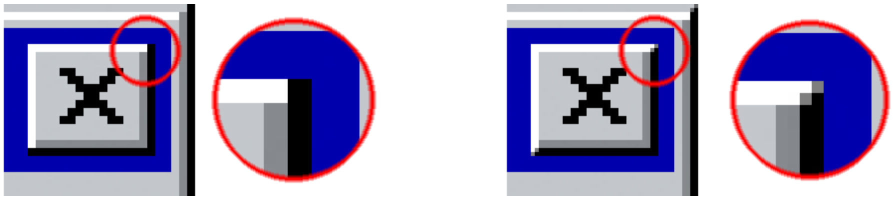
Why imitate when you can have the real thing?

There are people out there clever enough to have written a full x86-compatible CPU and hardware emulator… *in JavaScript*. It runs in your browser. **This one (https://copy.sh/v86/?profile=windows95)** works quite well, but I have a lot of appreciation for **this other one (https://win95.ajf.me/)**, a precariously assembled stack of software doing things it was never intended to. It's loading a Windows 95 disk image inside of a **DOSBox (https://www.dosbox.com/)** ported to **asm.js (http://asmjs.org/)**, a low-level subset of JavaScript optimized for running things that might run better anywhere than a browser. Perfect! With this at my fingertips, I could hoard an arsenal of screenshots to get my Windows 95 styling just right.

Pixels

As designers and front-end engineers, we debate over whether or not buttons should be flat or embossed, or whether they have rounded edges or not, but these are really choices about fashion. The button itself, as a concept, **has changed remarkably little in decades**

((https://vimeo.com/61556918)). Converting Tangram Play to a 20-year-old UI is akin to putting on a new outfit. But our authentic period costumes betray their modern manufacturing when you examine the seams. We can achieve a Windows 95 emboss effect via CSS, but look closer:



The upper-right corner of dialog boxes. Screenshot from Windows 95 on the left, browser implementation on the right.

On higher-resolution screens like a Mac OS X Retina monitor, it's impossible to get the vertices of these embossed edges to be a sharp, solid pixel. Instead, the corner connection is mitered for a softer transition. This makes sense if your borders are ever more than one pixel wide, a more common occurrence than mimicking Windows 95.[citation needed]

There are many more small details like this, some more egregiously defective than others. But you probably wouldn't notice any of it unless I pointed it out. Although we use the term "pixel-perfect" to describe the reskin, it's more like pixel-close-enough. Accuracy isn't a strict rule, but the point is to make the right moves where it matters.

Typeface

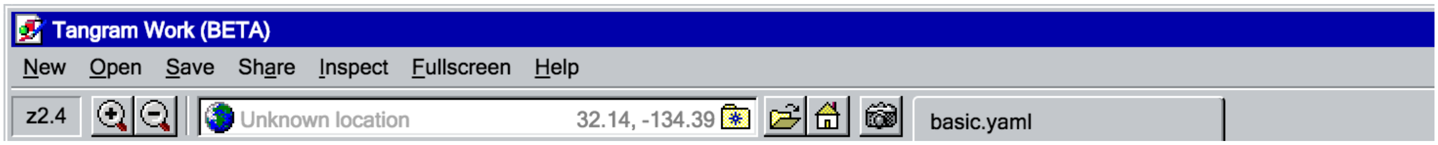
One area where it matters is in typography. Good typography makes the design, and bad typography destroys it. Choosing the right typeface for the job is one of the most important things a designer can do, and this is especially true for presenting historical fiction. Period films are especially prone to anachronisms. Choose poorly, and you **pull a typographer right out of the movie (http://www.marksimonson.com/notebook/view/typecasting)**.

The opening shot of Mission Impossible: Ghost Protocol (2011) is this beautiful aerial view of Budapest overlaid by Verdana, a "web-safe" font bundled in Windows and Mac OS X operating systems since 1996.

Thus, it was important to use the original Windows 95 typeface. It was easy to find out what it's called: a few minutes of investigation turned up "MS Sans Serif." But using it is another matter entirely.

These early fonts are called *bitmap fonts*, which meant that each pixel of the font was placed directly on a grid. This worked well when computer screens had low resolutions, but each time you needed a different size of font, a designer had to re-draw every pixel for every letter for each desired font size. As screen resolutions increased, and as designers' needs evolved, it became much more efficient to transition to *vector fonts*, where each shape of a letter can be specified by an equation describing a line. Nearly every font you encounter now will be a vector font, and this allows fonts to be scaled up or down nearly infinitely and look sharp and crisp at every size.

In 2000, Microsoft transitioned its own system fonts to a vector format. MS Sans Serif no longer existed, and it was replaced by Microsoft Sans Serif, vectorized and renamed. Although it's commonly available on mainstream operating systems, including Mac OS X, it looks too modern, too out of place, and it takes me right out of the movie.

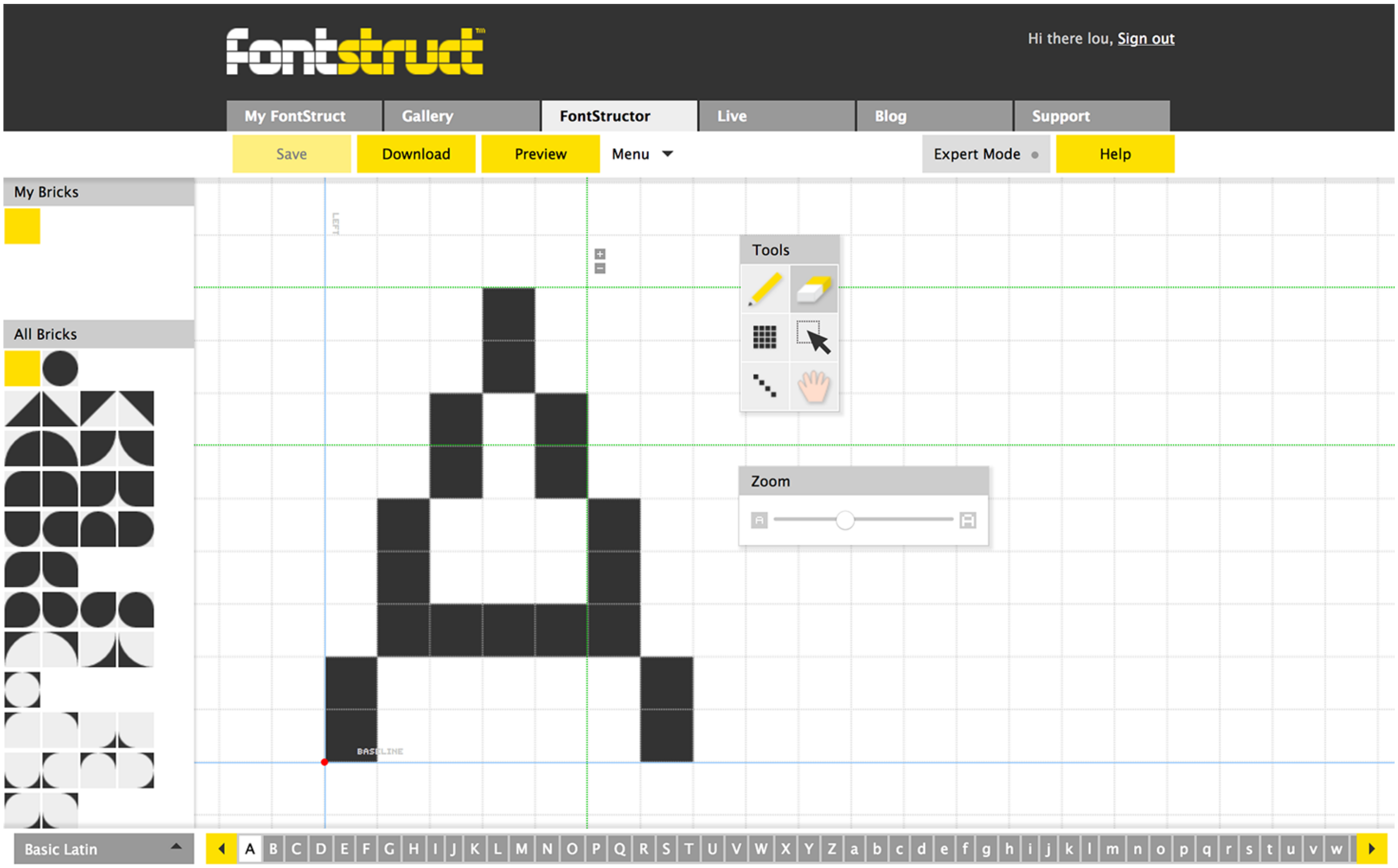


Could we have liberated the original font file (`SSERIFE.FON`) from a Windows 95 installation and used it directly in the application?

Unfortunately, no. I didn't actually have a working installation I could copy the file out of; furthermore, distributing a typeface you don't have a license to raises some legal questions. But the final nail in the coffin is that `.FON` font types are such an old format that they **don't appear to be supported anyway (https://bugs.chromium.org/p/chromium/issues/detail?id=23915)**.

I then learned of the excellent **Fontstruct (https://fontstruct.com/)** tool, which provides a way of creating your own fonts by assembling "tiles" of various shapes. By drawing with square tiles, I could simulate the pixels of a bitmap font, albeit in a modern format. It's a good thing I could easily create screenshots of every letter from the emulators, so I got to work.

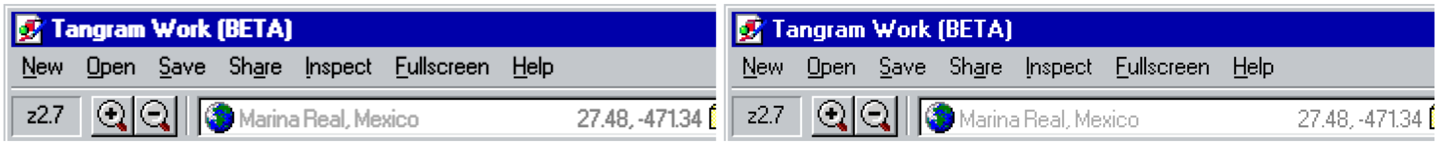
It only took about an afternoon to have both the regular weight and bold weight of my MS Sans Serif recreation ready and available in TrueType.


fontstruct
Hi there lou, Sign out
My FontStruct
Gallery
FontStructor
Live
Blog
Support
Save
Download
Preview
Menu
Expert Mode
Help
My Bricks
All Bricks
Tools
Zoom
BASELINE
Basic Latin
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w

Success! You can download these: ***MS Sans Serif (https://fontstruct.com/fontstructions/show/1384746/ms-sans-serif)*** *and* ***MS Sans Serif Bold (https://fontstruct.com/fontstructions/show/1384862/ms-sans-serif-bold)****.*

Since the font is really a vector font, Chrome is perfectly happy to scale it to whatever size you want it to be with anti-aliased edges. So I made sure that all the text in Tangram Work was using its "native" 9-point size (there were no glyphs made in any other point size), and finally, we use this CSS property to ensure that the edges aren't smoothed out:

```
body {
  -webkit-font-smoothing: none;
}
```


Tangram Work (BETA)
New
Open
Save
Share
Inspect
Fullscreen
Help
z2.7
Marina Real, Mexico
27.48, -471.34
Tangram Work (BETA)
New
Open
Save
Share
Inspect
Fullscreen
Help
z2.7
Marina Real, Mexico
27.48, -471.34

Removing anti-aliasing, before and after.

Scrollbars

Another detail I'm quite proud of are the scrollbars. Initially, I'd assumed that the scrollbars were native interface components of the *actual* operating system, and would have limited styling capacity. In Windows 95, there are arrow buttons at either end of the scrollbar, which don't exist in Mac OS X.

But it turns out that Chrome's layout engine offers complete controls of the scrollbar's appearance, including the parts of the scrollbar that aren't actually there. It's because they don't use the native scrollbars at all. They're actually using *secret HTML elements styled as native scrollbars.*

Wait, huh? How does that work?

Most front-end web developers are familiar with the concept of **Shadow DOM (https://glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/)**. To explain what that is, in a grossly oversimplified way (this isn't a technical paper), we'll first need to describe what an HTML element is. Nearly all HTML elements are simple boxes with some styling rules attached to them. You can put text in them, or you can put in other boxes (that is, other HTML elements), or any number of those in any combination. You can nest these boxes, like matryoshka dolls, as deep as you'd like. So a basic HTML snippet might look like this:

```
<div>
  <h1>Hi! I'm a heading!</h1>
</div>
```

This means that you have a `<div>` element, a basic box with no special properties, and inside of that you have `<h1>`, a box with some styles that makes the text larger and bolder. Inside the `<h1>` is some text. When a browser renders this HTML, it becomes a JavaScript object called the Document Object Model (or DOM for short). It's not a Shadow DOM yet, because what you see is exactly what you'll get.

But let's say I make a simple checkbox input: `<input type="checkbox" checked>`

☑

Try clicking on it to toggle it on or off; it's an actual checkbox. It's also as a "box" in HTML: one can style its borders or its background color, for example. Here's the thing: you see the container for the checkbox itself, but when it's checked, where does the glyph for the checkmark come from? We haven't put it inside the "box" in our HTML code, and it's not coming from our stylesheet, either.

That's Shadow DOM. Some HTML elements are "special" because they are actually much more complicated than a box with styles, so your browser has to "cheat" by secretly building it out of other elements nested inside of it that the real DOM doesn't know about. In fact, **most form elements (https://www.tjvantoll.com/2013/12/06/why-is-styling-form-elements-so-damn-hard/)** are constructed with one or more “shadow” elements. To access those elements, a front-end developer needs to know the magic incantations that enable her to cross the Shadow DOM “boundary,” but once she does, she can style the shadow elements inside using normal CSS. Unfortunately, not everything can be accessed in this way, and different browsers will often behave quite differently, because Shadow DOM hasn’t been a priority for web standards.

Scrollbars — at least in Chrome — is one example of Shadow DOM. Tangram Play already tweaks the appearance of scrollbars, so what I’d realized is that if you can style *some* parts of a scrollbar with CSS, then that must mean *none* of the scrollbar is actually a native operating system component. It would have to mean that Chrome implemented a generic scrollbar element that could be styled to look like *any* native scrollbar on each of its platforms.

Of course, there’s no actual `<scrollbar>` element — much less any existence of `<scrollbutton>` or what-have-you. But I already did know some of the magic incantations to get at the scrollbar’s secret elements, like `::-webkit-scrollbar-track`. How many more might there be, and could I gain control over all of them? **Thanks to CSS-Tricks, we have a spellbook for that (https://css-tricks.com/custom-scrollbars-in-webkit/)**.

With this knowledge in mind, we could absolutely achieve close-to-perfect Windows 95 scrollbars with stylesheets alone. Check out the live example here (warning: Chrome and Safari browsers only).

Finally, we had one more fun thing up our sleeve.

Globey, your new mapping companion

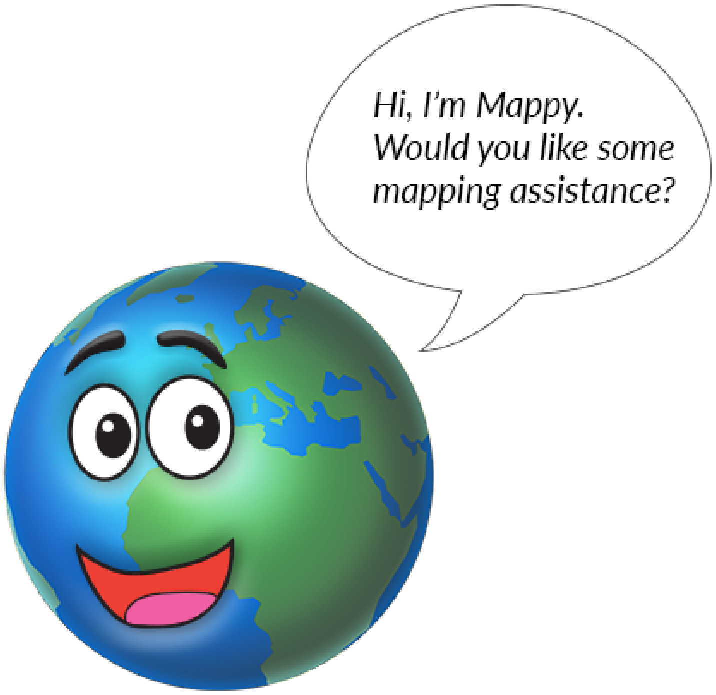
Katie here

Lou and I share a lot of things in common and our admiration for design made in a time without pixel perfect tools is massive. We might skew towards pure snark in 2017 on the 'retro' feel of these applications because we live in an age of **Creative Cloud (http://www.adobe.com/creativecloud.html)**, **Material Design (https://material.io/guidelines/)**, and the so-called "homogenization of web design", but a lot of the design that went into these tools were just as detailed and thoughtful (and even more so!) as current trends.

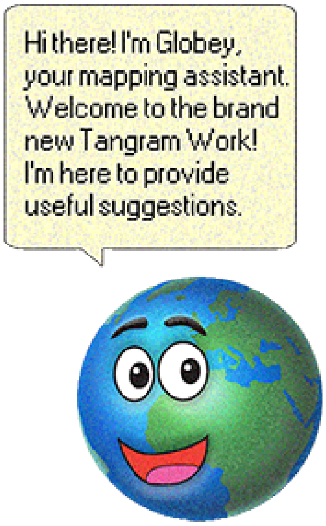
One of the most controversial additions in early UX development for Microsoft Office was the Jar Jar Binks of productivity platforms, **Clippy (https://en.wikipedia.org/wiki/Office_Assistant)**. Just try and mention Clippy nowadays and you get a lot of laughs. When I saw Lou demo this tremendous redesign of Tangram Play, I had to play my part and add a delightful helper, Globey.

Globey was completely built between Adobe Photoshop and Illustrator. I created an artboard full of inspiration for different aspects of Globey, ranging from Clippy (obviously), to **Map from Dora the Explorer (http://dora.wikia.com/wiki/Map)**, to the aesthetics of some of my favorite 90's video games like ***Where In The World Is Carmen Sandiego?* (https://en.wikipedia.org/wiki/Where_in_the_World_Is_Carmen_Sandiego%3F_(1996))** and ***Civilization II* (https://en.wikipedia.org/wiki/Civilization_II)**.

To make the globe, I made a dark blue circle in Photoshop and layered multiple circles with semi-opaque gradients in different blues to make it look 3D. This required a lot of playing with blending modes and making a couple different custom gradients. I popped into Google Earth and took a screenshot at the right view we wanted, with the world focused on Africa, Europe, and a bit of Asia. I popped that into Photoshop and scaled the image to the right size and free hand drew the continents with the pen tool. I also sketched a mouth, eyes and eyebrows in Illustrator, mostly based on Map from Dora the Explorer, and imported the face into Photoshop. All of this created a very nice looking globe. Tooooooo nice looking.


Hi, I’m Mappy.
Would you like some
mapping assistance?

I added some mosaic filtering to pixelate Globey and then resampled the colors being used in Photoshop by going to *Image* > *Mode* > *Indexed Color…* and changing the palette to be the Windows OS option. This merges all the layers so it took a lot of trial and error to get everything right. Once you do this, you can’t edit in Photoshop anymore with that color mode chosen, so I popped back into RGB mode and did a few last touches.

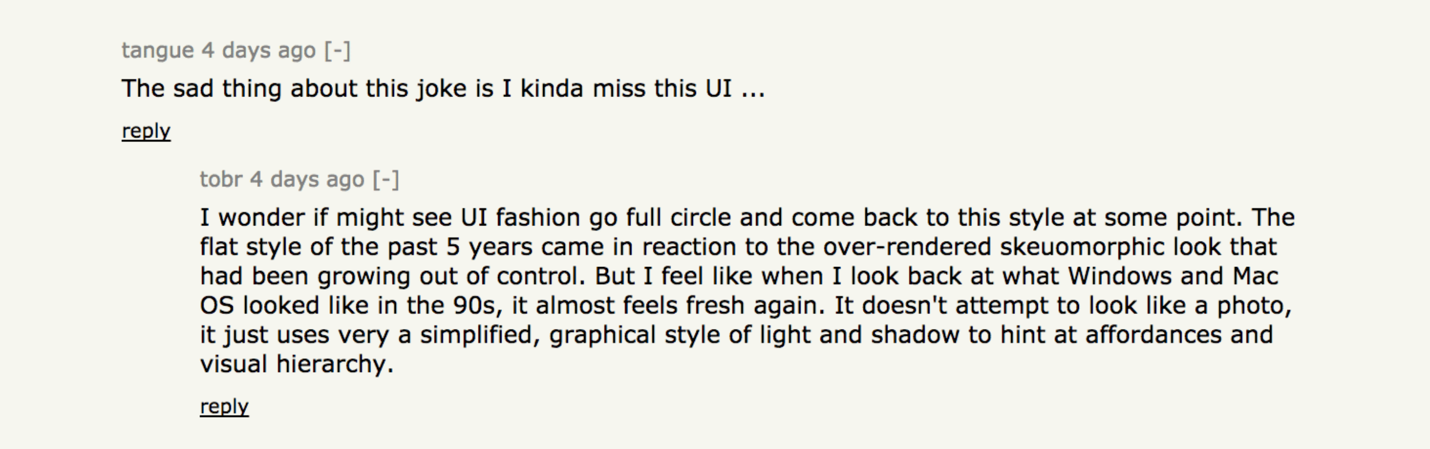

Hi there! I'm Globey, your mapping assistant. Welcome to the brand new Tangram Work! I'm here to provide useful suggestions.

Making animated GIFs in Photoshop is pretty simple, so to make Globey’s amazing eyebrow movement, I created two versions of Globey’s eyebrows and created two frames in the Timeline window, one with each different Globey. And, voila! Globey fits the “look” of a Windows 95 application and even can move a little. Lou popped Globey into Tangram Work and created an array to store various phrases for Globey to say.

Closing thoughts

Lou again

As Katie mentioned, it's easy to be tempted to poke some fun at the source material when you're making a parody of something old, as if saying, "look at how old and crappy this is!" But I have to stress again that exactly the opposite happened. As I recreated each component of Windows 95, I developed an enormous amount of respect for what the designers and engineers at Microsoft accomplished twenty years ago.



tangue 4 days ago [-]

The sad thing about this joke is I kinda miss this UI ...

reply

tobr 4 days ago [-]

I wonder if might see UI fashion go full circle and come back to this style at some point. The flat style of the past 5 years came in reaction to the over-rendered skeuomorphic look that had been growing out of control. But I feel like when I look back at what Windows and Mac OS looked like in the 90s, it almost feels fresh again. It doesn't attempt to look like a photo, it just uses very a simplified, graphical style of light and shadow to hint at affordances and visual hierarchy.

reply

Maybe we're onto something... (via ***Hacker News (https://news.ycombinator.com/item?id=14012224)****)*

At its core, Windows 95 is an extremely usable UI language. As I unearthed more reference material from old screenshots scattered throughout the web, I'm reminded that no era has universally bad design. Sure, there may be prevalent examples of poor choices from the 90's, but they don't diminish the excellent fundamentals that they extend from, except only in my memory.

As for Electron: we really like it. We'll likely explore this path more in the months to come. And lastly, we're so happy that Globey is a huge hit and is starting to garner a small fan base. Even though April Fool's Day has passed, Globey will remain a part of Tangram Play from now on, as a hidden **Easter Egg (https://en.wikipedia.org/wiki/List_of_Easter_eggs_in_Microsoft_products)**. We won't say how you trigger it — half the fun is in the discovery!

We would also like to thank Jessica Lord for Electron assistance and reviewing this blog post, Marcin Wichary for pointing us to Fontstruct, John Oram for additional contributions to Tangram Work, and finally our other colleagues at Mapzen for their assistance with Tangram Work QA and testing.

· 12 April 2017 ·

Lou Huang

Lou is a protagonist of an open world.

Katie Kowalsky

Katie is a mapslainer who knows how to make a proper cheese plate and is a serial to-do list maker.

© 2017 Mapzen

The world is weird and wonderful!

boundaryissues (/tag/boundaryissues) whosonfirst (/tag/whosonfirst) data (/tag/data)

TL;DR: we've added a CSV import feature to ***Boundary Issues (https://mapzen.com/blog/boundary-issues-properties/)****! Our web-based GeoJSON editor for Who's On First is not available for public use (yet), but we're still publicly documenting our progress out in the open. The more technical stuff with screengrabs comes after a bit of introductory context.*

The territory is itself weird

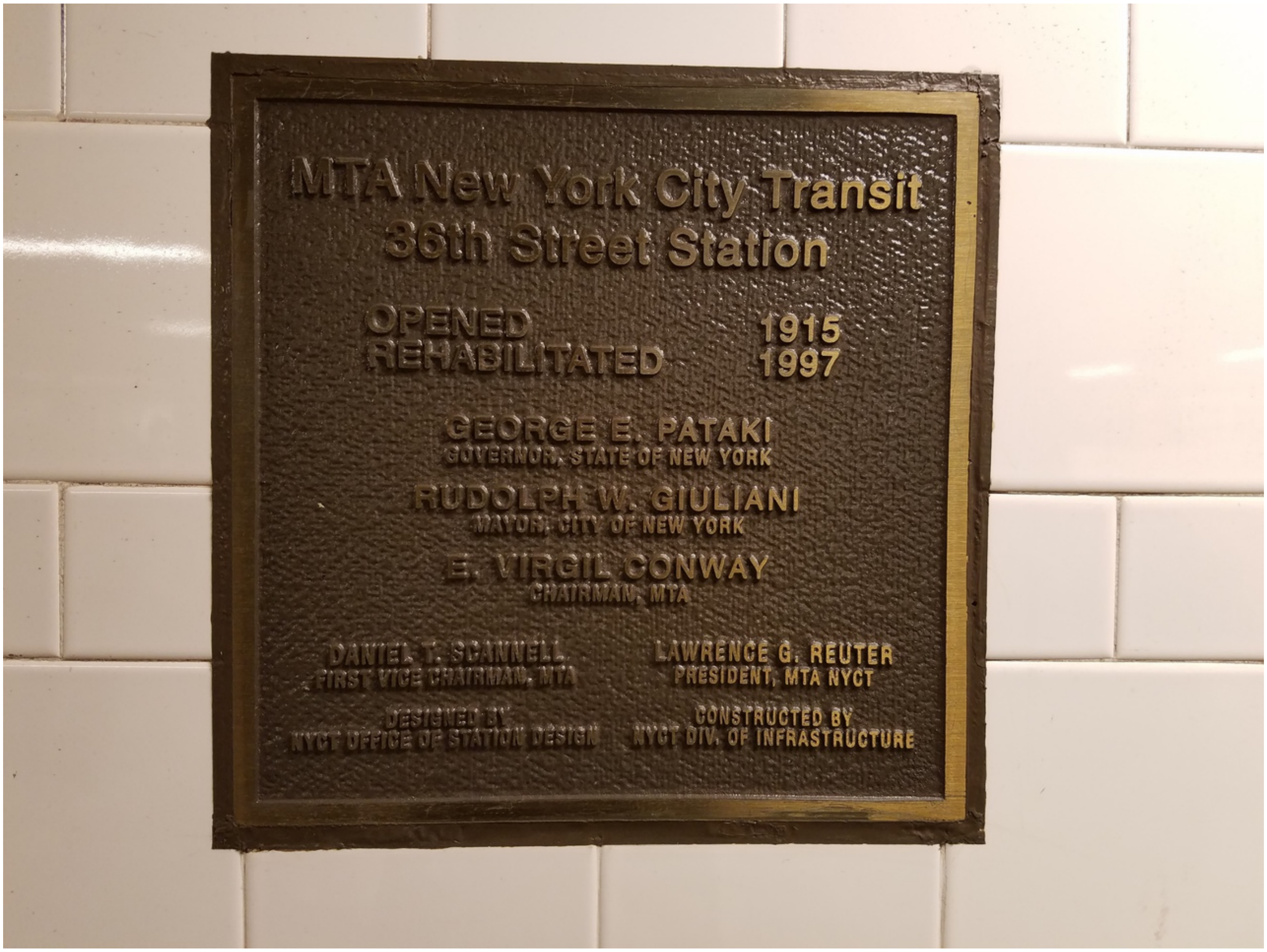
Like many New Yorkers who grew up on the west coast, the subway system is endlessly fascinating to me. Our sprawling 24-hour transit is a source of pride, the subject of gripes and strongly held opinions, and maybe the most interesting venue for **contemporary dance (http://gothamist.com/2015/06/29/two_showtime_dancers_give_us_the_lo.php)** in a city well known for its performing arts.

Naturally, we love to map the subway here in NYC. There's the familiar and practical **MTA official map (http://web.mta.info/maps/submap.html)** (whose **evolution (http://gizmodo.com/18-subway-maps-that-trace-nycs-transit-history-1242514280)** is also a subject of obsession) and its many **opinionated variations (http://www.citylab.com/design/2014/07/a-vignelli-inspired-map-designed-to-make-the-least-amount-of-people-mad/374680/)**. Two of my favorites are the **prewalker (http://www.urbandictionary.com/define.php?term=prewalking)**'s essential tool, **ExitStrategy (http://gizmodo.com/5310818/nyc-exit-strategy-the-other-nyc-subway-app-you-need)**, and **BulletMap (https://medium.com/@anthonydenaro/mapping-nyc-transit-all-of-it-e16e76a95a0e)**, a map optimized for transfers between the subway and bus networks.

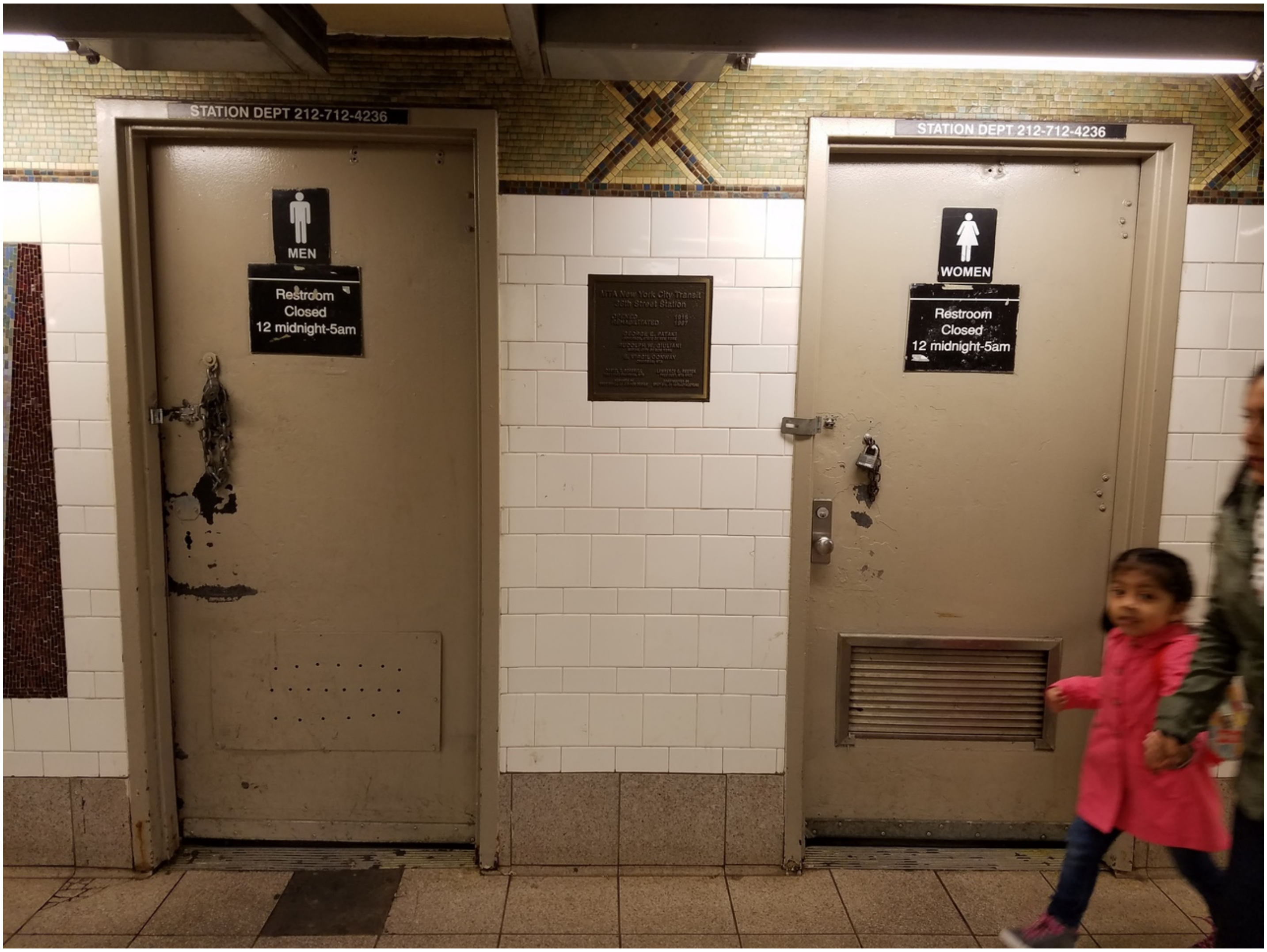
All of this is to say, the "subway map" means different things to different people. And that is because the subway itself means different things, depending on whether or not you're **able-bodied (https://www.dnainfo.com/new-york/20150608/bushwick/map-subway-map-showing-only-wheelchair-accessible-stations)**, just looking to **catch a show (http://nypost.com/2014/04/17/nyc-buskers-too-many-zooz-become-latest-internet-sensation/)**, or stocking up on **fashionionable collectables (http://www.theverge.com/2017/2/20/14674604/mta-supreme-metro-cards-nyc-subway-resale)**.

Zooming in

Let’s isolate ourselves to just one subway station, let’s say the **36th Street D/N/R/W stop (https://en.wikipedia.org/wiki/36th_Street_(BMT_Fourth_Avenue_Line))** in the historically working class neighborhood of Sunset Park. You can find this useful metadata embedded within the station itself.

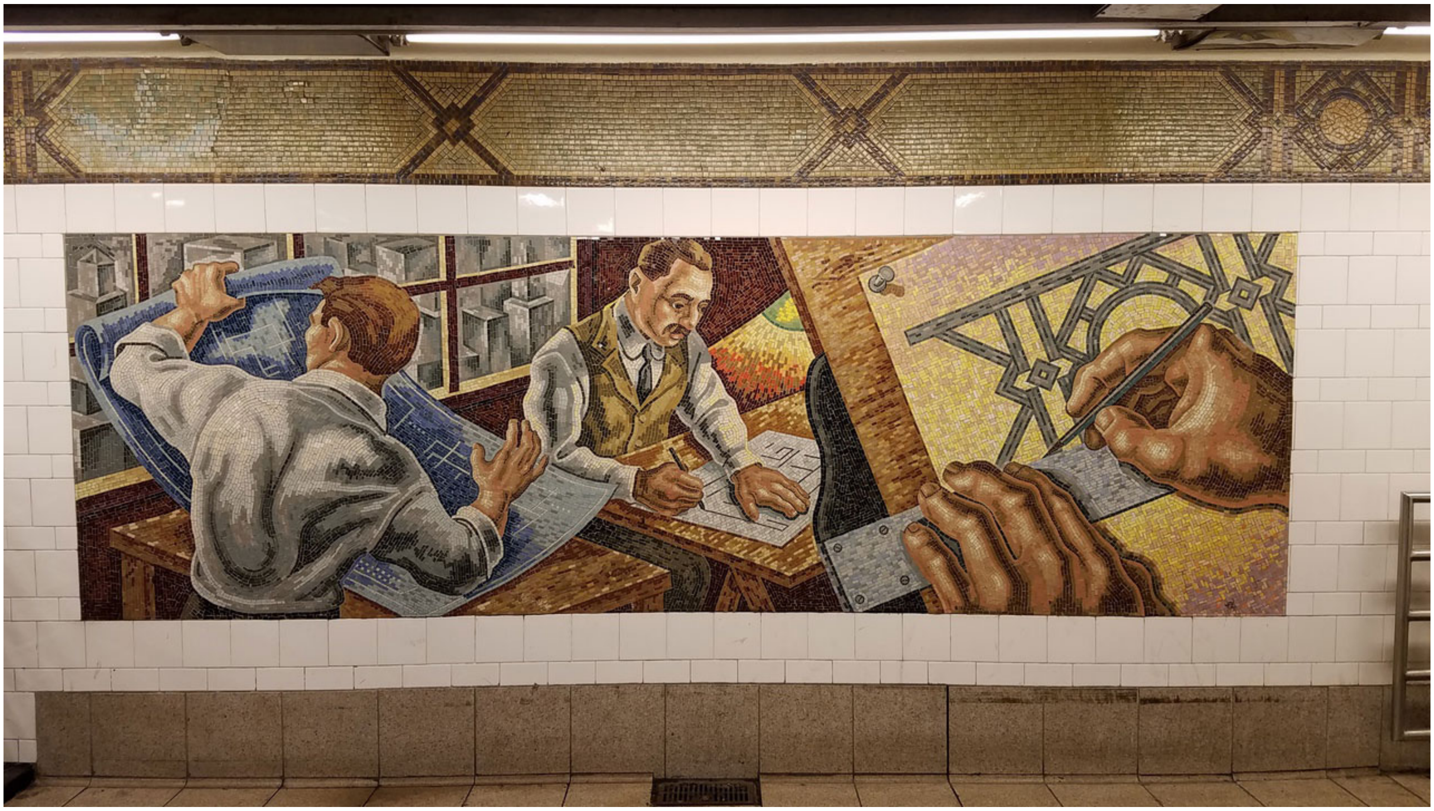

MTA New York City Transit
36th Street Station
OPENED 1915
REHABILITATED 1997
GEORGE E. PATAKI
GOVERNOR, STATE OF NEW YORK
RUDOLPH W. GIULIANI
MAYOR, CITY OF NEW YORK
E. VIRGIL CONWAY
CHAIRMAN, MTA
DANIEL T. SCANNELL
FIRST VICE CHAIRMAN, MTA
LAWRENCE G. REUTER
PRESIDENT, MTA NYCT
DESIGNED BY
NYCT OFFICE OF STATION DESIGN
CONSTRUCTED BY
NYCT DIV. OF INFRASTRUCTURE

Let’s just stand back a little bit. Behold, there are restrooms here! (For those not familiar with NYC subway facilities, this is uncommon.)

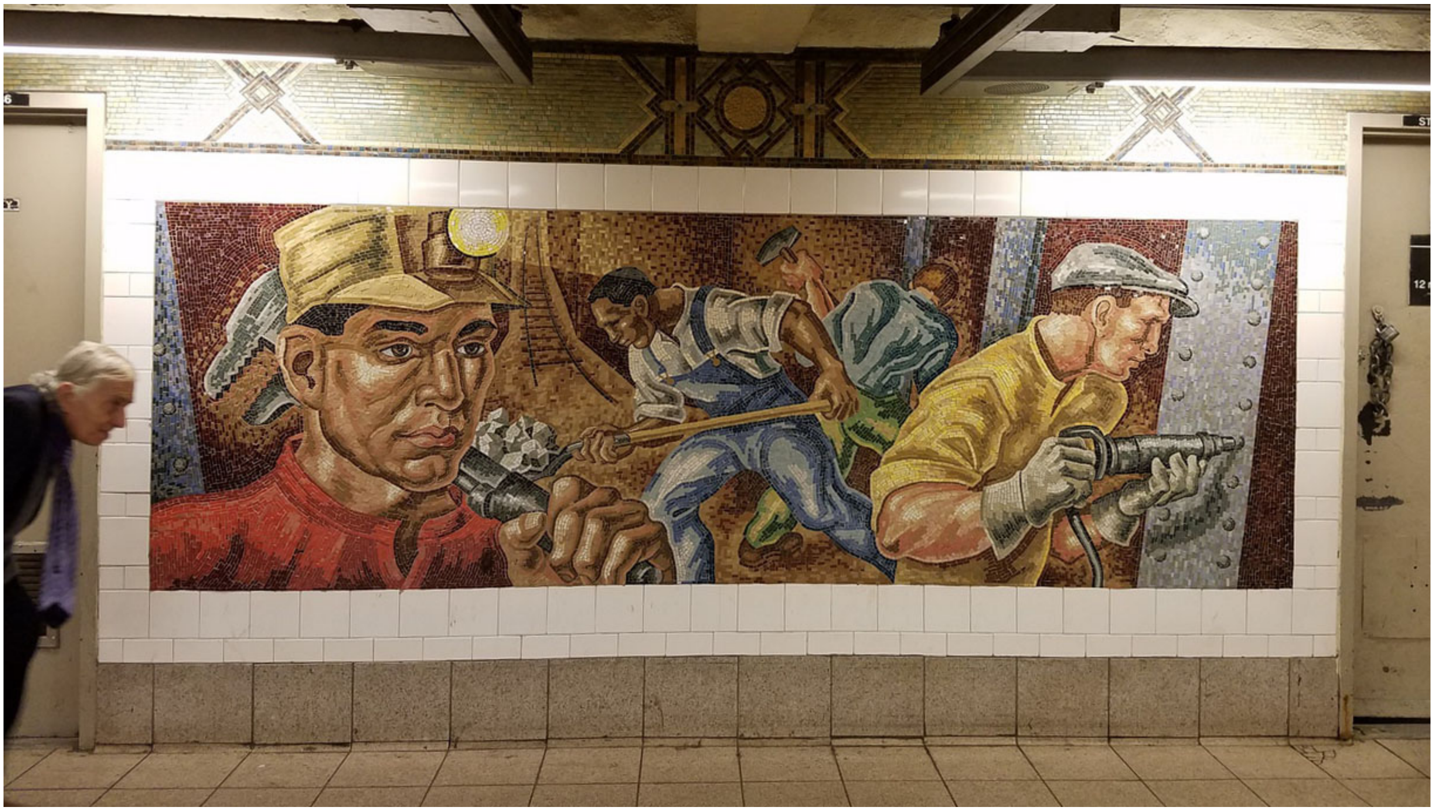

STATION DEPT 212-712-4236
MEN
Restroom Closed 12 midnight-5am
STATION DEPT 212-712-4236
WOMEN
Restroom Closed 12 midnight-5am

Also included are more valuable hours-of-operation metadata, especially useful if you find yourself returning home from a late night out, looking for relief. (Either rush home *before midnight* or take your time and wait until *after 5am*.)

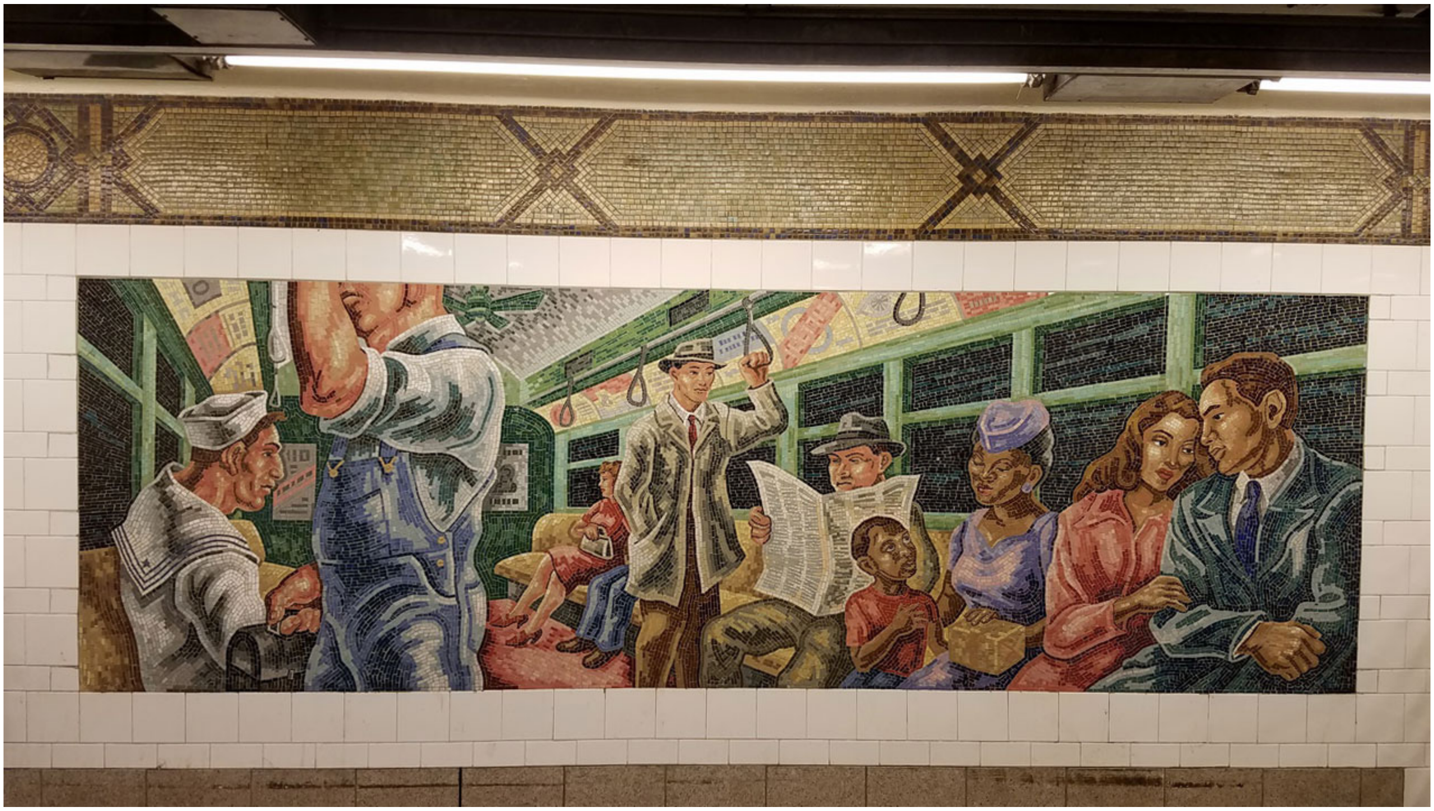
Let’s keep looking around this station. Oh man, check out these tile murals!



The designers. (Hover or tap on this if you want to blow your mind.)



The builders.

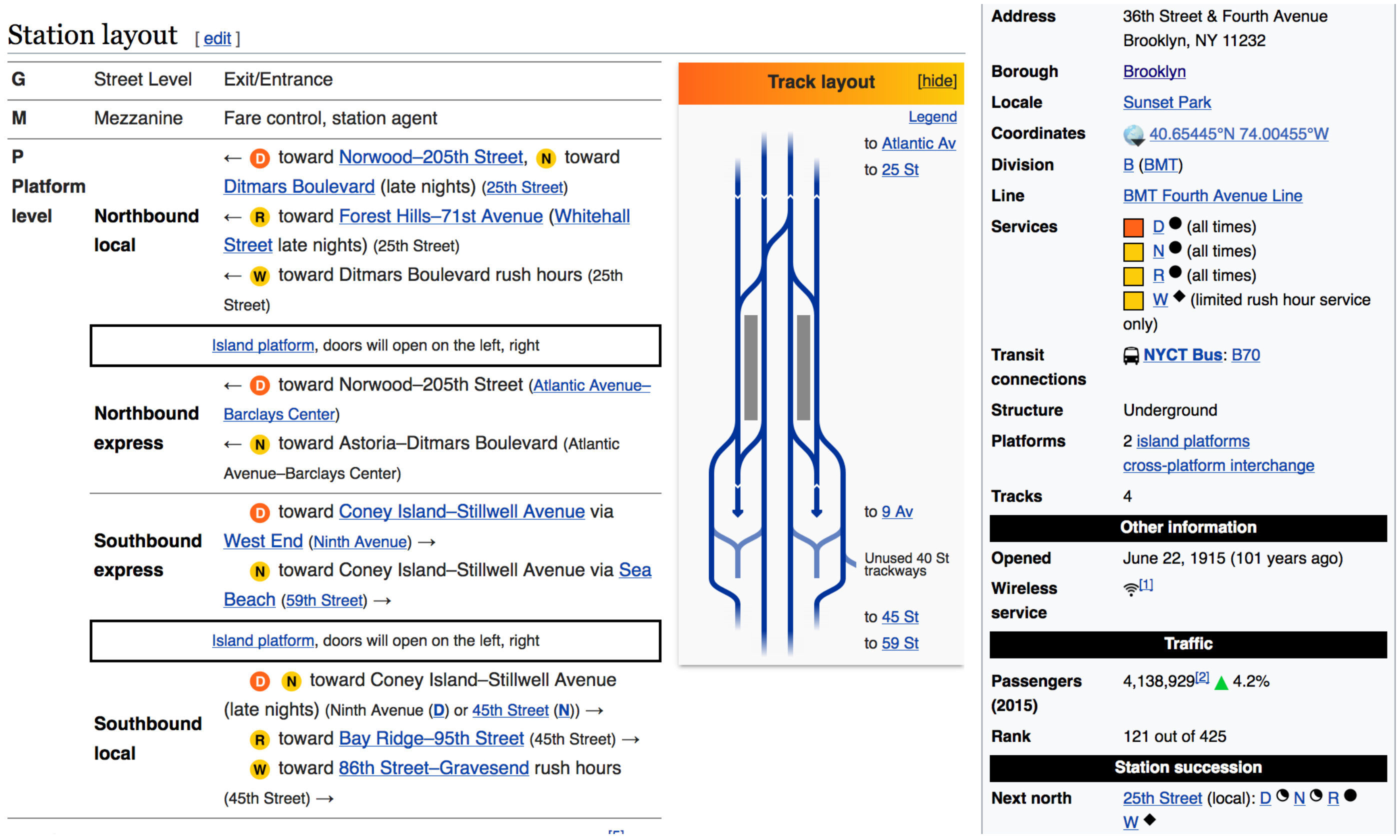


The riders.

Like all of the public art in NYC subway stations, we've got more good metadata here. (Shout out to Peter Columbo, whose name is misspelled on the plaque, but whose mosaic handiwork is **also at 110th St and Lexington (http://www.metmuseum.org/exhibitions/listings/2012/byzantium-and-islam/blog/cultural-connections/posts/byzantium-and-hip-hop)**.)

Owen Smith
An Underground Movement:
Designers, Builders, Riders
1998
Ceramic mosaic
Fabricated by Peter Colombo Artistic Mosaics
Commissioned by Metropolitan Transportation Authority
Arts for Transit and MTA New York City Transit

Checking out the **Wikipedia article (https://en.wikipedia.org/wiki/36th_Street_(BMT_Fourth_Avenue_Line))** for the station gives us an even denser picture of the station's attributes. I am as fascinated by the train enthusiasts who obsessively compile this info, as I am by the train systems themselves. (n.b. I very much **count myself (https://www.youtube.com/playlist?list=PL_WssN5hKWUY9exh9UonSJBO9ntpL137Y)** as one of those weirdo train enthusiasts.)



Station layout [edit]

G	Street Level	Exit/Entrance
M	Mezzanine	Fare control, station agent
P Platform level	Northbound local	← D toward Norwood–205th Street, N toward Ditmars Boulevard (late nights) (25th Street) ← R toward Forest Hills–71st Avenue (Whitehall Street late nights) (25th Street) ← W toward Ditmars Boulevard rush hours (25th Street)
	Island platform, doors will open on the left, right	
	Northbound express	← D toward Norwood–205th Street (Atlantic Avenue–Barclays Center) ← N toward Astoria–Ditmars Boulevard (Atlantic Avenue–Barclays Center)
	Southbound express	D toward Coney Island–Stillwell Avenue via West End (Ninth Avenue) → N toward Coney Island–Stillwell Avenue via Sea Beach (59th Street) →
	Island platform, doors will open on the left, right	
	Southbound local	D N toward Coney Island–Stillwell Avenue (late nights) (Ninth Avenue (D) or 45th Street (N)) → R toward Bay Ridge–95th Street (45th Street) → W toward 86th Street–Gravesend rush hours (45th Street) →

Track layout [hide]

Address	36th Street & Fourth Avenue Brooklyn, NY 11232
Borough	Brooklyn
Locale	Sunset Park
Coordinates	40.65445°N 74.00455°W
Division	B (BMT)
Line	BMT Fourth Avenue Line
Services	🟧 D ● (all times) 🟨 N ● (all times) 🟨 R ● (all times) 🟨 W ◆ (limited rush hour service only)
Transit connections	**NYCT Bus**: B70
Structure	Underground
Platforms	2 island platforms cross-platform interchange
Tracks	4
Other information	
Opened	June 22, 1915 (101 years ago)
Wireless service	[1]
Traffic	
Passengers (2015)	4,138,929[2] ▲ 4.2%
Rank	121 out of 425
Station succession	
Next north	25th Street (local): D ◐ N ◑ R ● W ◆

Despite the impressive diagram illustrating how the station's unused tracks are positioned, Wikipedia does not mention restrooms or public art. That there are restrooms, that the mural here is a great example of Social Realism, are facts that are *just as valid* and frankly *more interesting* than the latitude and longitude for this place. Lots of the nuance that captures why we should even care about places tends to get ignored by datasets that are considered comprehensive.

(There was once **an app for finding NYC subway art (http://www.wnyc.org/story/286899-an-app-for-finding-nyc-subway-art/)**, but it appears to have been discontinued.)

Zooming out

Taking our previous example at 36th Street, we could imagine systematically capturing the kinds and locations of public art in transit centers (we could also do the same for restrooms and other amenities).

This being a post on the Mapzen blog, I'm interested in how that information can make its way into our open gazetteer, **Who's On First (https://whosonfirst.mapzen.com/)**. It's important to caveat that this type of transit information has *not yet* been gathered and associated with our place records, but it could be. We have all the tools in place, and an extensible structure that can capture new forms of place nuance.

The trick is in how we encode the properties for each place record, and in particular how we keep those property names from conflicting with each other. As I wrote in my last blog post about **Boundary Issues and property editing (https://mapzen.com/blog/boundary-issues-properties/)**:

> Instead of specifying a `tags` property, we use `wof:tags`, meaning it's a part of the Who's On First namespace. This lets us keep things that are common to *all WOF records*, like `wof:name`, separate from from Mapzen *opinionated things* like `mz:categories`

The namespaces for WOF properties act as a cue about the nature of each property: is it "general purpose" or is it an "opinionated assertion?" It also serves another purpose: *where did this opinion originate from?* If you see a property with `sg:tags`, that `sg:` prefix lets you know that the data came from **SimpleGeo (https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces)**.

This system allows us to keep a property like `wof:name` (the name of the place, "36th Street") distinct from `public_art:name` (the mural's title, "An Underground Movement: Designers, Builders, Riders"). The namespaces assigns a kind of credit/blame for where a given detail originated from. We maintain a GitHub repository, **whosonfirst-sources (https://github.com/whosonfirst/whosonfirst-sources/)**, that describes who is responsible for which properties.

(Quick side note: just to be especially clear, `public_art:name` is a fictional property name chosen for the purpose of illustrating the logic of prefixes and sources. Who's On First has **an existing convention (https://whosonfirst.mapzen.com/spelunker/concordances/sfac/#10/37.8166/-122.5057)** of treating public art as a *first-class place*, rather than a set of properties that decorate wherever an artwork physically manifests itself. A helpful test for answering the question "is it a place?" is: *can I ask someone to meet me there?* As in, "let's meet over by **Fearless Girl (https://gregfallis.com/2017/04/14/seriously-the-guy-has-a-point/)**.")

```
{
    "id": 404734187,
    "fullname": "Yahoo! GeoPlanet (formerly Where On Earth)",
    "name": "whereonearth",
    "prefix": "woe",
    "key":"id",
    "url": "http://developer.yahoo.com/geo/geoplanet/",
    "license": "http://developer.yahoo.com/geo/geoplanet/data/"
}
```

A Who's On First *source* is composed of a short list of attributes like `name`, `fullname`, `prefix`, `url` encoded into a publicly accessible JSON file. We maintain a single **list of all the sources (https://github.com/whosonfirst/whosonfirst-sources/blob/master/sources/README.md#simplegeo)**, including **The New York Times (https://github.com/whosonfirst/whosonfirst-sources/blob/master/sources/nytimes.json)** (prefix `nyt`), **United States Census Bureau (https://github.com/whosonfirst/whosonfirst-sources/blob/master/sources/nytimes.json)** (prefix `uscensus`), and our own **Burrito Justice (https://github.com/whosonfirst/whosonfirst-sources/blob/master/sources/burritojustice.json)**, whose microhood, **La Lengua (https://burritojustice.com/la-lengua/)**, has gone on to become **standardized (https://bernalwood.com/2015/02/05/extremely-graphic-infographic-visualizes-2014-bernal-heights-microhood-real-estate-trends/)** in other mapping taxonomies.

The range of ways we might describe a place will differ vastly depending on interests and needs. Using property prefixes gives us a useful way to group those needs, avoid naming conflicts, and let them all cohabitate within a single WOF record.

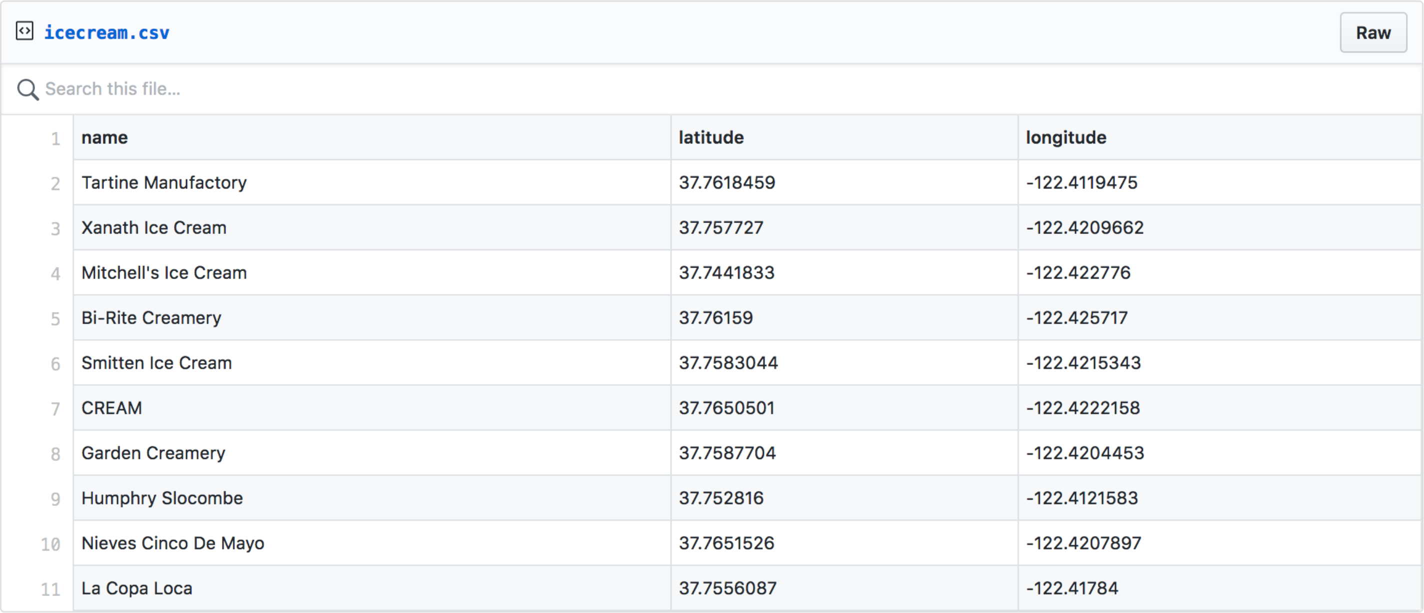
Who's On First: it's not just data

One of our self-imposed design constraints is that we don't get to choose your dev stack. This is why we publish everything as GeoJSON flat files instead of, say, a PostgreSQL dump file. If you use PostgreSQL/PostGIS, you'll find a way to get the data into your tables.

For my work on **Boundary Issues (https://github.com/whosonfirst/whosonfirst-www-boundaryissues)**, our bespoke web-based content management system for WOF records, my Mapzen colleagues are the primary customer. However, they are not the only customer, and we are just starting to pull in outside contributors. (If you want to be a guinea pig, please **get in touch (boundaryissues@mapzen.com)**!). A lot of my recent focus working on Boundary Issues has been about lowering the bar for contributing data.

WOF ❤️ CSV

While **GeoJSON (http://geojson.org/)** offers an amazing level of flexibility and compatibility for describing places, it turns out CSV is an *even better* format for expressing many places at once. First of all it's just been around longer; **FORTRAN 77 supported it in 1978 (https://en.wikipedia.org/wiki/Comma-separated_values#History)**. That longevity means it has support in a wide range of spreadsheet editors, from VisiCalc to Google Sheets.



name	latitude	longitude
Tartine Manufactory	37.7618459	-122.4119475
Xanath Ice Cream	37.757727	-122.4209662
Mitchell's Ice Cream	37.7441833	-122.422776
Bi-Rite Creamery	37.76159	-122.425717
Smitten Ice Cream	37.7583044	-122.4215343
CREAM	37.7650501	-122.4222158
Garden Creamery	37.7587704	-122.4204453
Humphry Slocombe	37.752816	-122.4121583
Nieves Cinco De Mayo	37.7651526	-122.4207897
La Copa Loca	37.7556087	-122.41784

Unlike JSON (or XML), data encoded into CSV format doesn't require that you read the entire file before you can start parsing it. That means CSV files can get *really really big*, and your software will still be able to read a single row efficiently. There are drawbacks to CSV as well, chiefly among them is it's harder to express nested information inside its columnar structure. It's all a trade-off, and choosing the right data format to encode stuff in has everything to do with the circumstances at hand.

Six sections into the post and…

That brings us to the lede, buried under quite a bit of context and exposition: *Boundary Issues now supports importing CSV files* 🎉. This is important and not just for the convenience of editing WOF data outside of specialized geo-software (**QGIS**

(http://www.qgis.org/en/site/forusers/download.html) I love you, but your user interface is bringing me down). CSV support means Boundary Issues can now import more data into Who's On First, from a wider range of sources. And more importantly, we can invite more people to edit within spreadsheet software they already have at hand, increasing the number of perspectives describing WOF places.

The world is wonderfully multifaceted

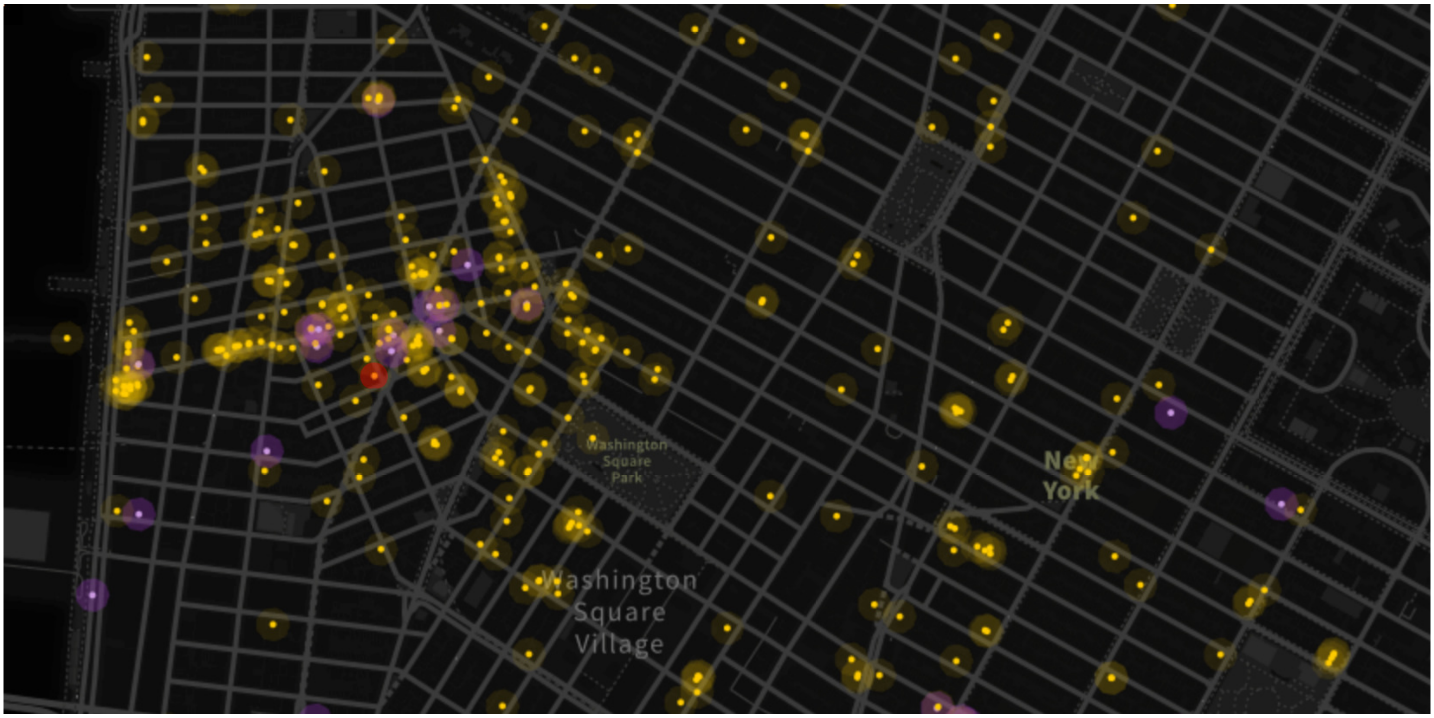
The territory means **different things to different people (https://twitter.com/Walldo/status/852040146446831616)**. Depending on your perspective, the kinds of data that are captured about places may be missing, insufficient, or downright hostile. Who's On First is *opinionated*—like all datasets, no collection is truly unbiased—but we hope to be aware of when we're asserting our own opinions about places and create a framework where a polyglot of *place-feels* will be welcome.

The multifaceted maps we make simply reflect the weird and wonderful territory they represent.

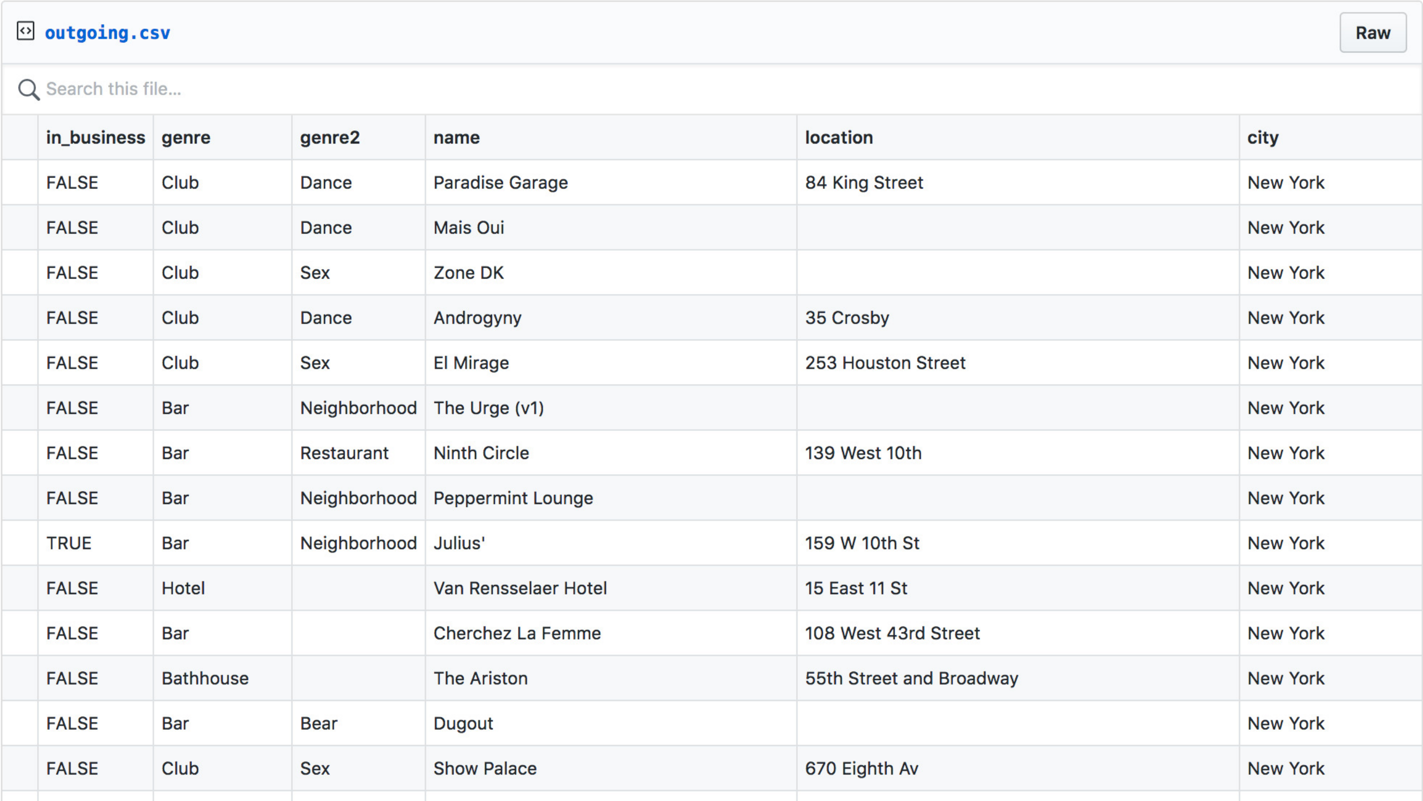
Like **the Internet (https://peoplecentered.net/the-internet-is-made-of-people/)**, Who's On First data is *made of people*. Those who have the capability to edit our raw data necessarily reflect themselves in those edits. They are present in the data through choices about which places get edited, and the types of nuanced assertions they make about places.

Throughout the development of the CSV import feature, I had the pleasure of working with Jeff Ferzoco's dataset for his excellent historic mapping project, **OUTgoing (http://outgoingnyc.com/)**.

> OUTgoing: The Hidden History of New York's Gay Nightlife—is an effort to map documented historic locations of gay nightlife in New York City as far back and comprehensively as possible. We now have all 50 states with legalized gay marriage, and the LGBT experience is in the midst of a 21st century upshift. It's time to look back and get some perspective and data—in order to look forward. From historic records, we've gathered 969 nighttime spots—bars, clubs, bathhouses, coffee houses, cruising areas, late-night diners, etc.—from the 1800s to 2015. It is nowhere near complete.

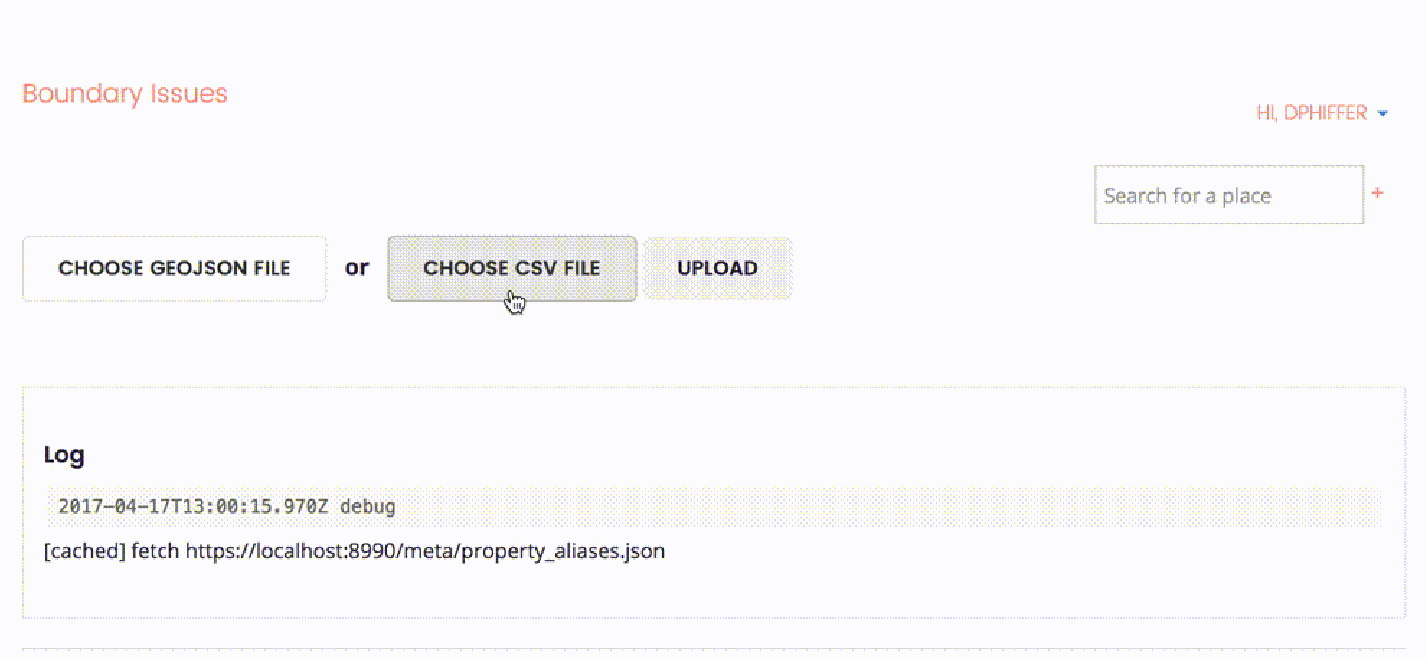


Jeff was kind enough to share a CSV file of all his collected places, which is dense with excellent metadata like `year_opened` , `year_closed` , and `mafia_owned` .



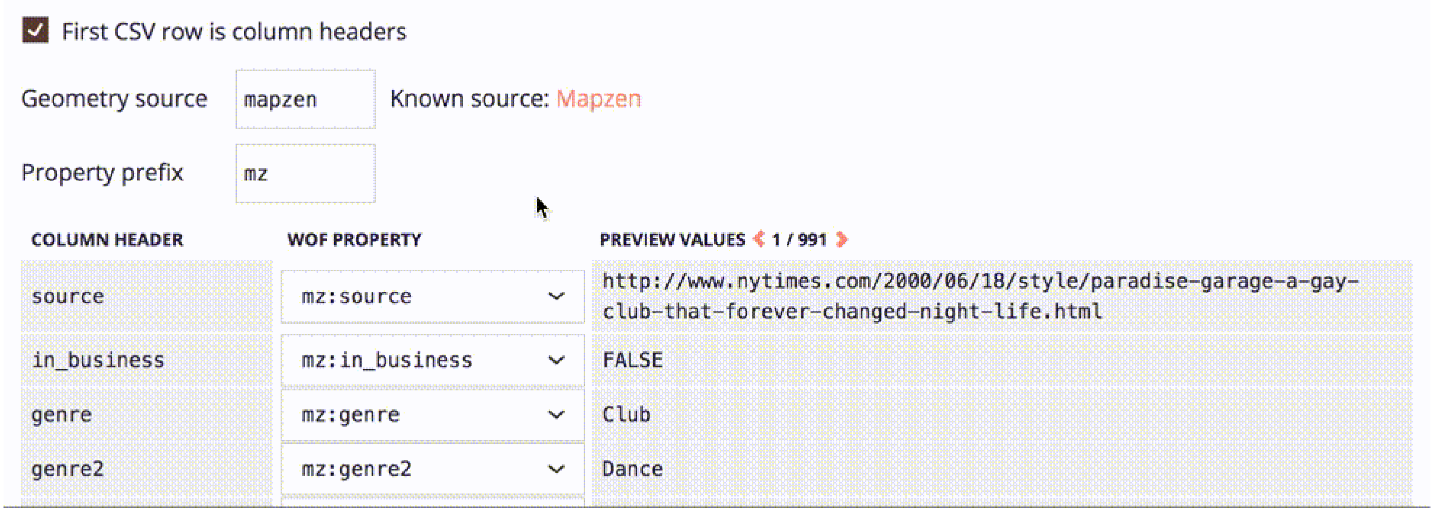
in_business	genre	genre2	name	location	city
FALSE	Club	Dance	Paradise Garage	84 King Street	New York
FALSE	Club	Dance	Mais Oui		New York
FALSE	Club	Sex	Zone DK		New York
FALSE	Club	Dance	Androgyny	35 Crosby	New York
FALSE	Club	Sex	El Mirage	253 Houston Street	New York
FALSE	Bar	Neighborhood	The Urge (v1)		New York
FALSE	Bar	Restaurant	Ninth Circle	139 West 10th	New York
FALSE	Bar	Neighborhood	Peppermint Lounge		New York
TRUE	Bar	Neighborhood	Julius'	159 W 10th St	New York
FALSE	Hotel		Van Rensselaer Hotel	15 East 11 St	New York
FALSE	Bar		Cherchez La Femme	108 West 43rd Street	New York
FALSE	Bathhouse		The Ariston	55th Street and Broadway	New York
FALSE	Bar	Bear	Dugout		New York
FALSE	Club	Sex	Show Palace	670 Eighth Av	New York

Here’s how we can start importing that data into Who’s On First: select the CSV file we want to upload.



```
2017-04-17T13:00:15.970Z debug
[cached] fetch https://localhost:8990/meta/property_aliases.json
```

The upload interface reveals some general options, allowing us to specify whether the CSV file has a column header row, identify a *source* for the data (in this case, **`outgoing` (https://github.com/whosonfirst/whosonfirst-sources/tree/master/sources#outgoing)**), and choose a *prefix* for the properties we are merging into Who's On First.



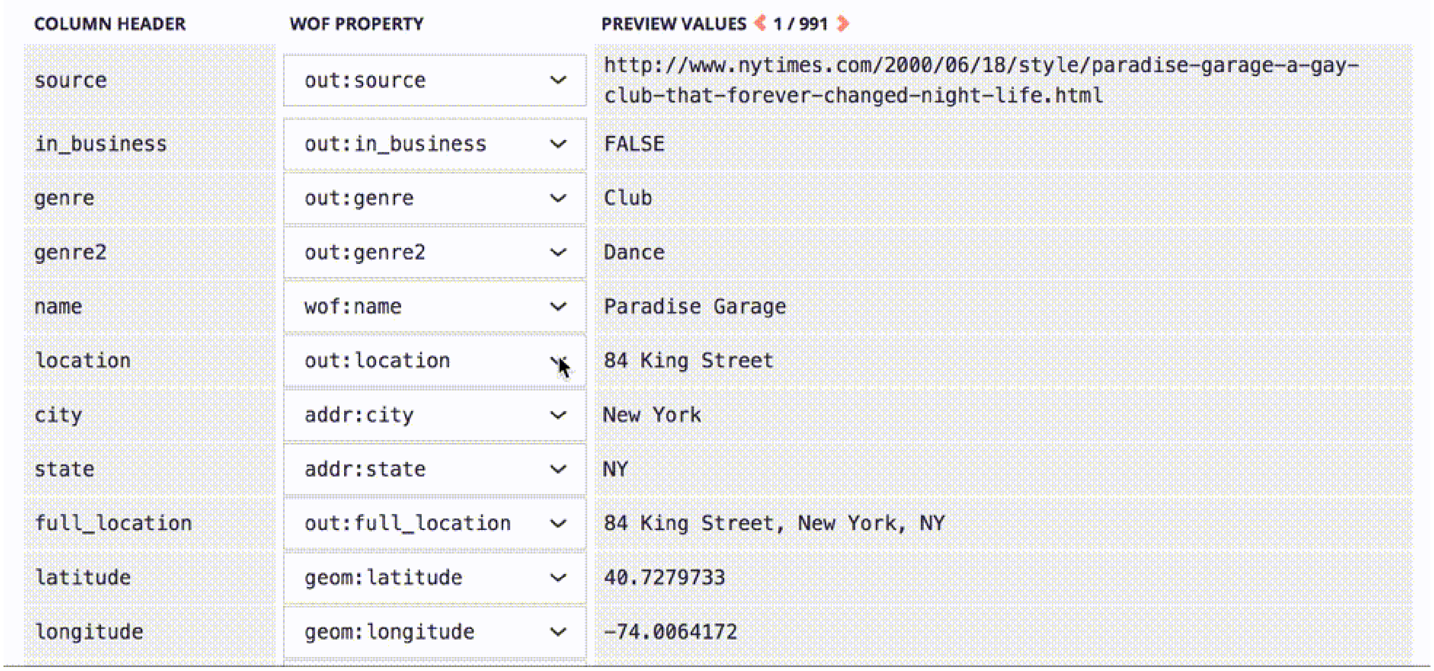
☑ First CSV row is column headers

Geometry source: mapzen — Known source: Mapzen

Property prefix: mz

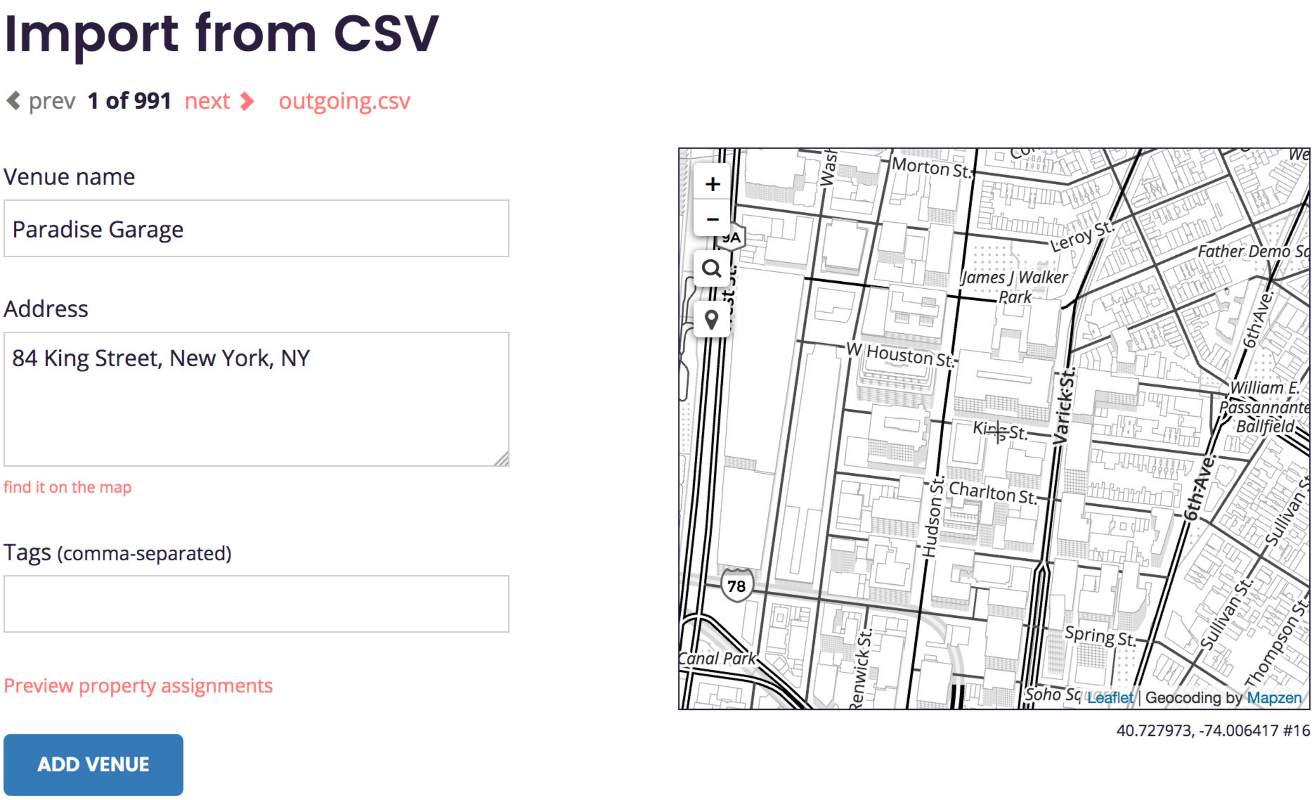
COLUMN HEADER	WOF PROPERTY	PREVIEW VALUES ‹ 1 / 991 ›
source	mz:source	http://www.nytimes.com/2000/06/18/style/paradise-garage-a-gay-club-that-forever-changed-night-life.html
in_business	mz:in_business	FALSE
genre	mz:genre	Club
genre2	mz:genre2	Dance

Next we will map *which columns* should be saved to *which properties*. Some columns map onto generic, well-known WOF properties like `wof:name` , `geom:latitude` , and `geom:longitude` , while others get saved as new properties with the `out:` prefix. The code detects predictable column headings like `latitude` and `longitude` and maps them automatically.



COLUMN HEADER	WOF PROPERTY	PREVIEW VALUES ❮ 1 / 991 ❯
source	out:source	http://www.nytimes.com/2000/06/18/style/paradise-garage-a-gay-club-that-forever-changed-night-life.html
in_business	out:in_business	FALSE
genre	out:genre	Club
genre2	out:genre2	Dance
name	wof:name	Paradise Garage
location	out:location	84 King Street
city	addr:city	New York
state	addr:state	NY
full_location	out:full_location	84 King Street, New York, NY
latitude	geom:latitude	40.7279733
longitude	geom:longitude	-74.0064172

Once that property mapping is ready, we can click the “upload” button and begin stepping through each of the rows in the CSV file.



Import from CSV

❮ prev **1 of 991** next ❯ outgoing.csv

Venue name: Paradise Garage

Address: 84 King Street, New York, NY

find it on the map

Tags (comma-separated):

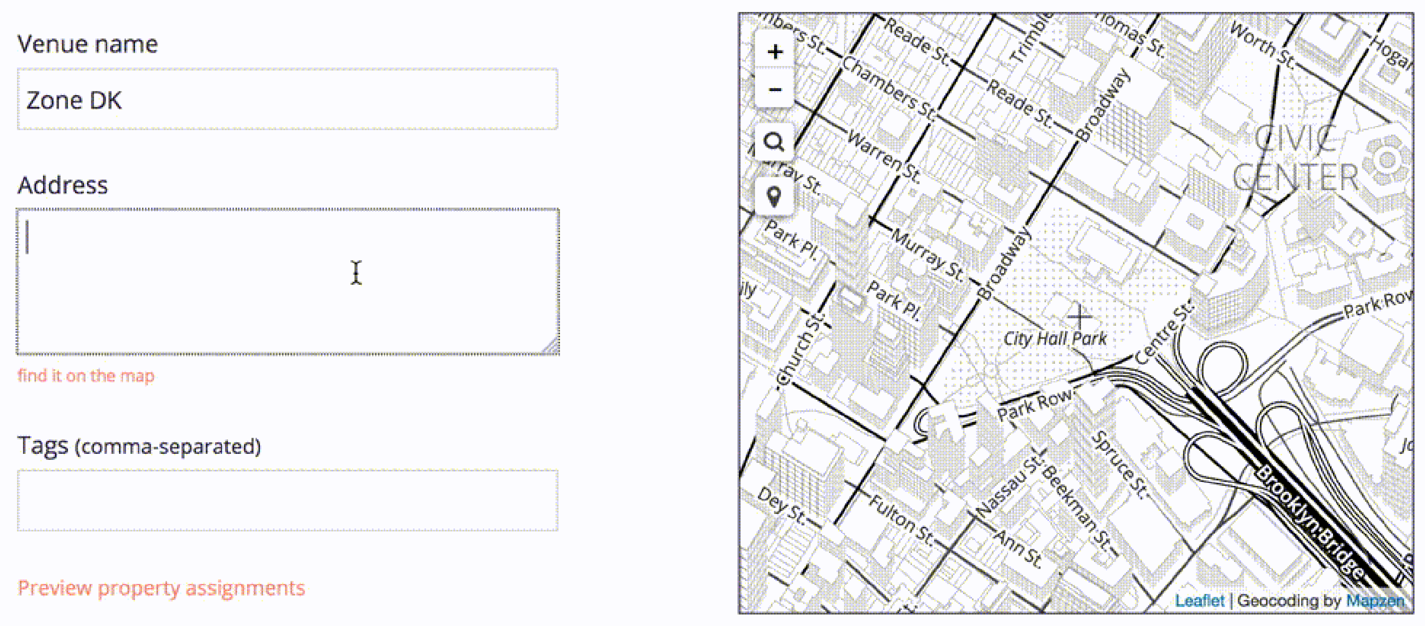
Preview property assignments

ADD VENUE

40.727973, -74.006417 #16

+
−
Morton St.
Leroy St.
James J Walker Park
Father Demo Sq
6th Ave.
W Houston St.
Varick St.
William E. Passannante Ballfield
King St.
Charlton St.
Hudson St.
6th Ave.
Sullivan St.
78
Spring St.
Sullivan St.
Thompson St.
Canal Park
Renwick St.
Soho Sq
Leaflet | Geocoding by Mapzen

This row-by-row process allows us to make any necessary adjustments to the record before we import it into the Who’s On First dataset. In some cases, we may need to choose which address is the correct one.



Venue name

Zone DK

Address

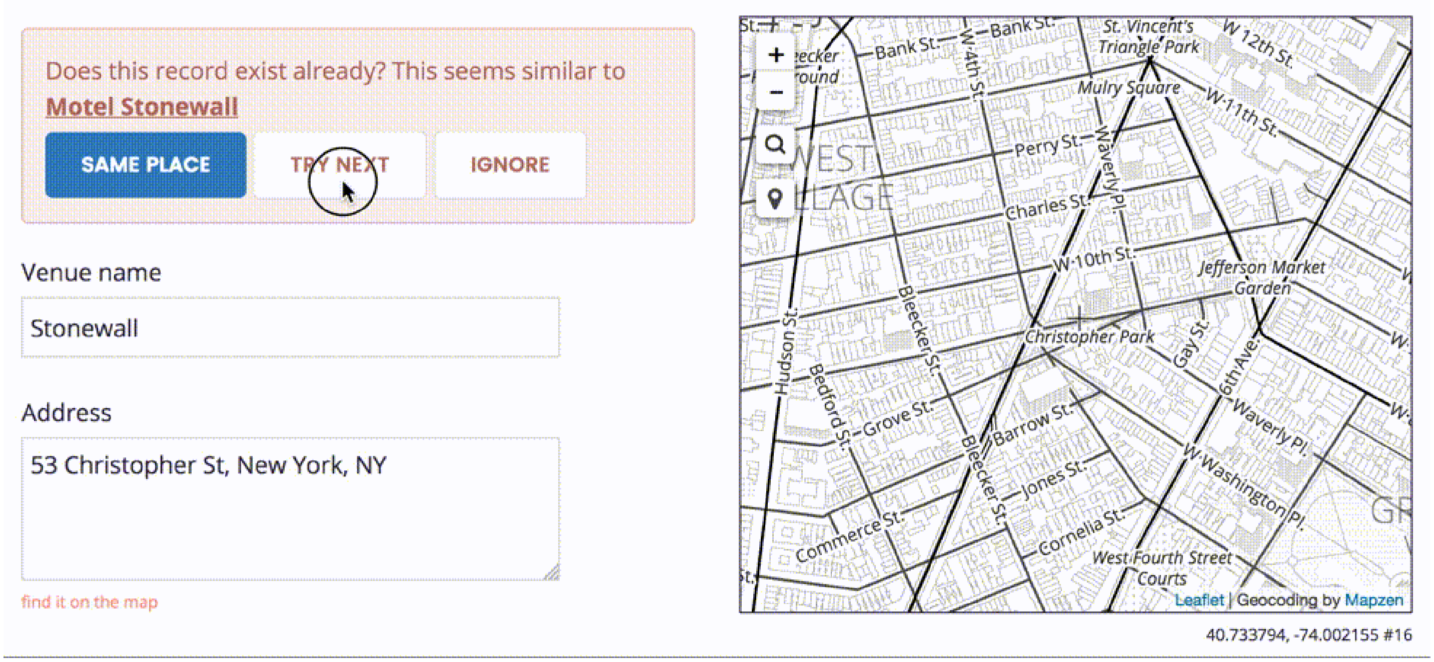
find it on the map

Tags (comma-separated)

Preview property assignments

+
−
Reade St.
Chambers St.
Warren St.
Murray St.
Park Pl.
Broadway
Trimble
Thomas St.
Worth St.
CIVIC CENTER
City Hall Park
Centre St.
Park Row
Church St.
Dey St.
Fulton St.
Nassau St.
Beekman St.
Spruce St.
Ann St.
Brooklyn Bridge
Leaflet | Geocoding by Mapzen

We also want to avoid duplicating existing records, if possible.



Does this record exist already? This seems similar to **Motel Stonewall**

SAME PLACE
TRY NEXT
IGNORE

Venue name

Stonewall

Address

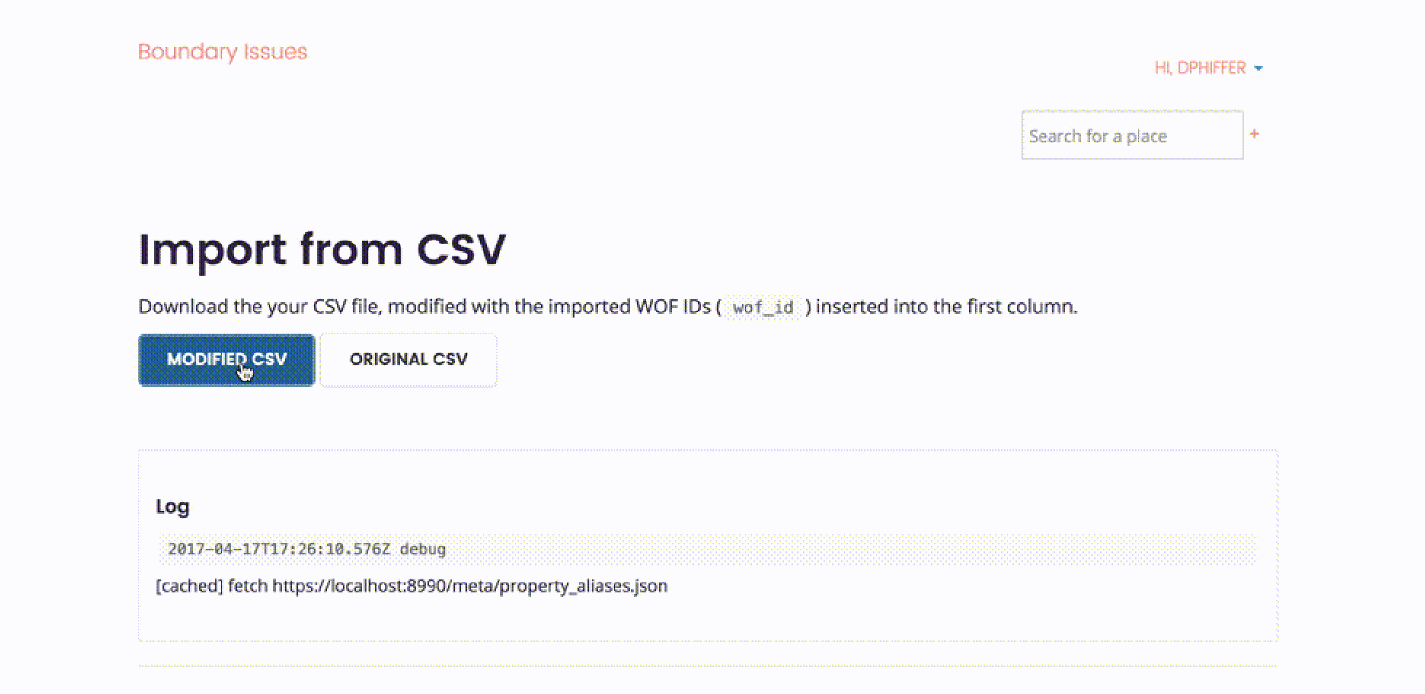
53 Christopher St, New York, NY

find it on the map

+
−
Bank St.
W 4th St.
St. Vincent's Triangle Park
W 12th St.
Mulry Square
W 11th St.
WEST VILLAGE
Perry St.
Waverly Pl.
Charles St.
W 10th St.
Jefferson Market Garden
Hudson St.
Bleecker St.
Christopher Park
Gay St.
6th Ave.
Bedford St.
Grove St.
Barrow St.
Jones St.
W Washington Pl.
Commerce St.
Cornelia St.
West Fourth Street Courts
Leaflet | Geocoding by Mapzen

40.733794, -74.002155 #16

Once we complete the step-through import process, we can download a new CSV file modified to include a new `wof_id` column. This additional column allows us to round-trip our edits back into Who's On First, effectively turning your spreadsheet editor into a new external interface to change properties en masse.



Import from CSV

Download the your CSV file, modified with the imported WOF IDs (`wof_id`) inserted into the first column.

Log

```
2017-04-17T17:26:10.576Z debug
[cached] fetch https://localhost:8990/meta/property_aliases.json
```

Working with real data has been extremely helpful for shaping the CSV step-through feature. Huge thanks to **Jeff (https://twitter.com/zingbot)** for sharing his research and for being so generous with the underlying dataset. The OUTgoing data is quirky and opinionated, and was instrumental in identifying unexpected edge cases as I was writing the code.

We **look forward to meeting (mailto:boundaryissues@mapzen.com?subject=CSV%20FTW)** your weird and wonderful CSVs!

Excuse me while I promote some things here

Plug #1: *This blog post was adapted into a talk at **csv,conf (https://csvconf.com/)** on May 3 in Portland, OR. Thanks for coming by! **Here are the slides (https://phiffer.org/etc/csvconf.pdf)**, and the **slides + video (https://phiffer.org/etc/csvconf.zip)** if you missed it.*

Plug #2: *We are extending the deadline for our **call for internship applications (https://whosonfirst.mapzen.com/interns/#2017)**. Please apply if you found this discussion interesting, and want to help us improve how we we present Who's On First to the world.*

· 17 April 2017 ·

Dan Phiffer

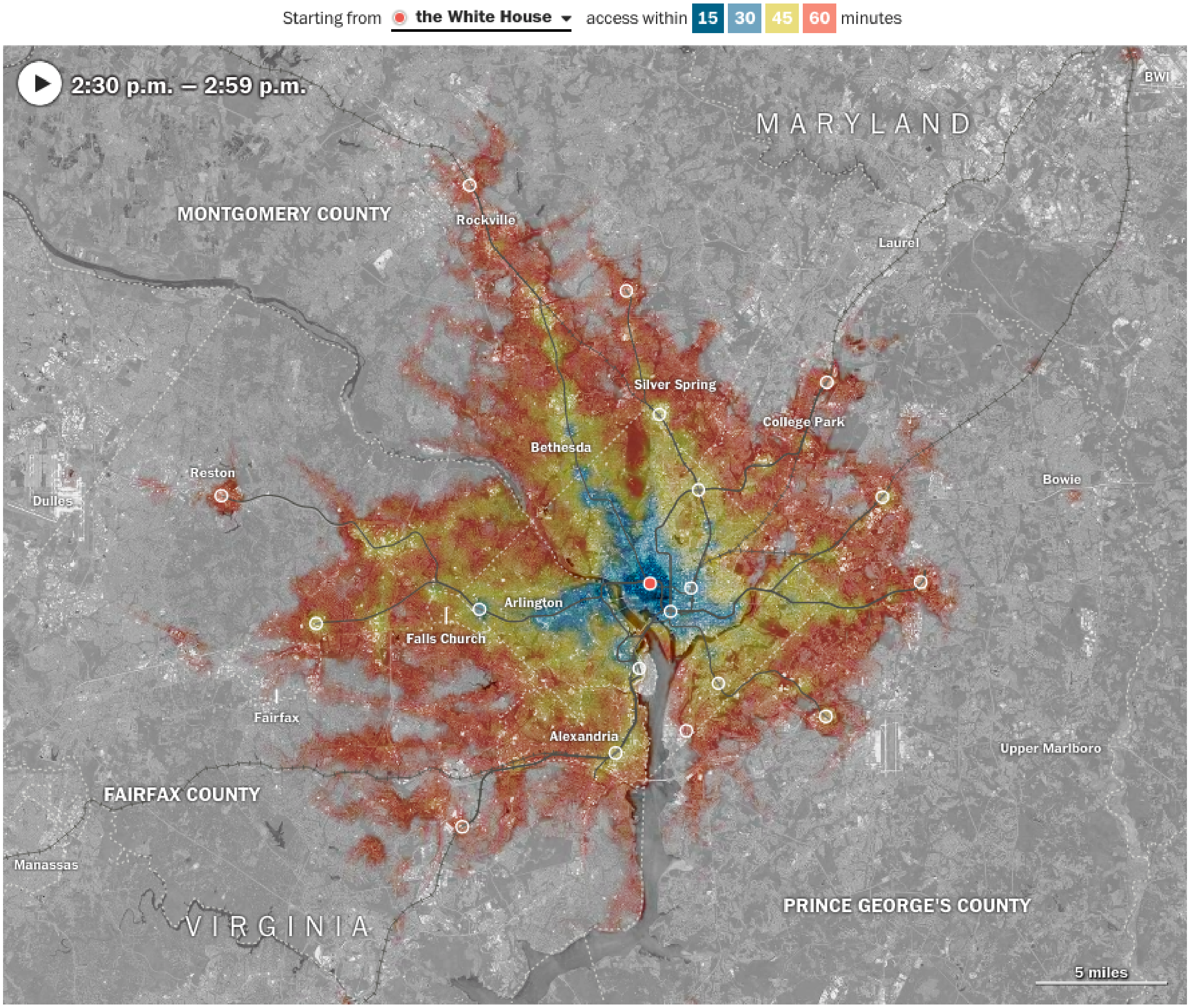
Dan is a data carpenter, which tbh is kind of a made up title. But he likes it all the same.

© 2017 Mapzen

Putting isochrones to work

mobility (/tag/mobility)

John Muyskens, Armand Emamdjomeh and Faiz Siddiqui of the Washington Post (https://www.washingtonpost.com/graphics/local/transit-access/?utm_term=.92e436b10eac) used the **Mapzen Mobility isochrone service (https://mapzen.com/documentation/mobility/isochrone/api-reference/)** to estimate commute times in the DC area. They used area the **include and exclude features (https://mapzen.com/blog/exclusion/)** to show how planned transit service reductions will have a heavier impact on lower-income communities.


Starting from ● the White House ▾ access within 15 30 45 60 minutes
2:30 p.m. – 2:59 p.m.
BWI
MARYLAND
MONTGOMERY COUNTY
Rockville
Laurel
Silver Spring
College Park
Bethesda
Reston
Dulles
Bowie
Arlington
Falls Church
Fairfax
Alexandria
Upper Marlboro
FAIRFAX COUNTY
Manassas
VIRGINIA
PRINCE GEORGE'S COUNTY
5 miles

Note: The composite map combines the accessible areas for departure times between 5:30 p.m. and 5:59 p.m. Source: Mapzen Mobility Explorer, Transitland

“Here’s where D.C. public transit can take you — and who gets left behind” **(https://www.washingtonpost.com/graphics/local/transit-access/?utm_term=.9fe6a9ab46d5)**

> *..the analysis showed that some of the region’s least-accessible areas also are the most transit-dependent. Northeast and Southeast Washington contain census tracts where more than 40 percent of households are both transit-reliant and lower-income, the only such corridor in the region.*

How did they do this?

> *The Mobility Explorer allows choosing the mode of transit and filtering out transit routes from use in generating isochrones. The rush-hour and late-night isochrones use buses and trains, as well as walking. A second set of late-night isochrones were created, filtering out Metro service.*

You can read more on the inequity of distance and commutes in the DC area in their feature article, "**When commuting in the D.C. region, distance doesn't tell the whole story (https://www.washingtonpost.com/local/trafficandcommuting/when-commuting-in-the-dc-region-distance-doesnt-tell-the-whole-story/2017/04/15/b466fdb6-1ef1-11e7-a0a7-8b2a45e3dc84_story.html?utm_term=.6143e6975def)**".

Editing Isochrones

Transportation planners often require tools to analyze and predict the impact of potential changes to the transportation network. These tools are often equipped to handle "what-if" conditions. For example, what is the impact of adding a new road or transit line or what is the impact of removing a transit line, closing a bridge for maintenance?

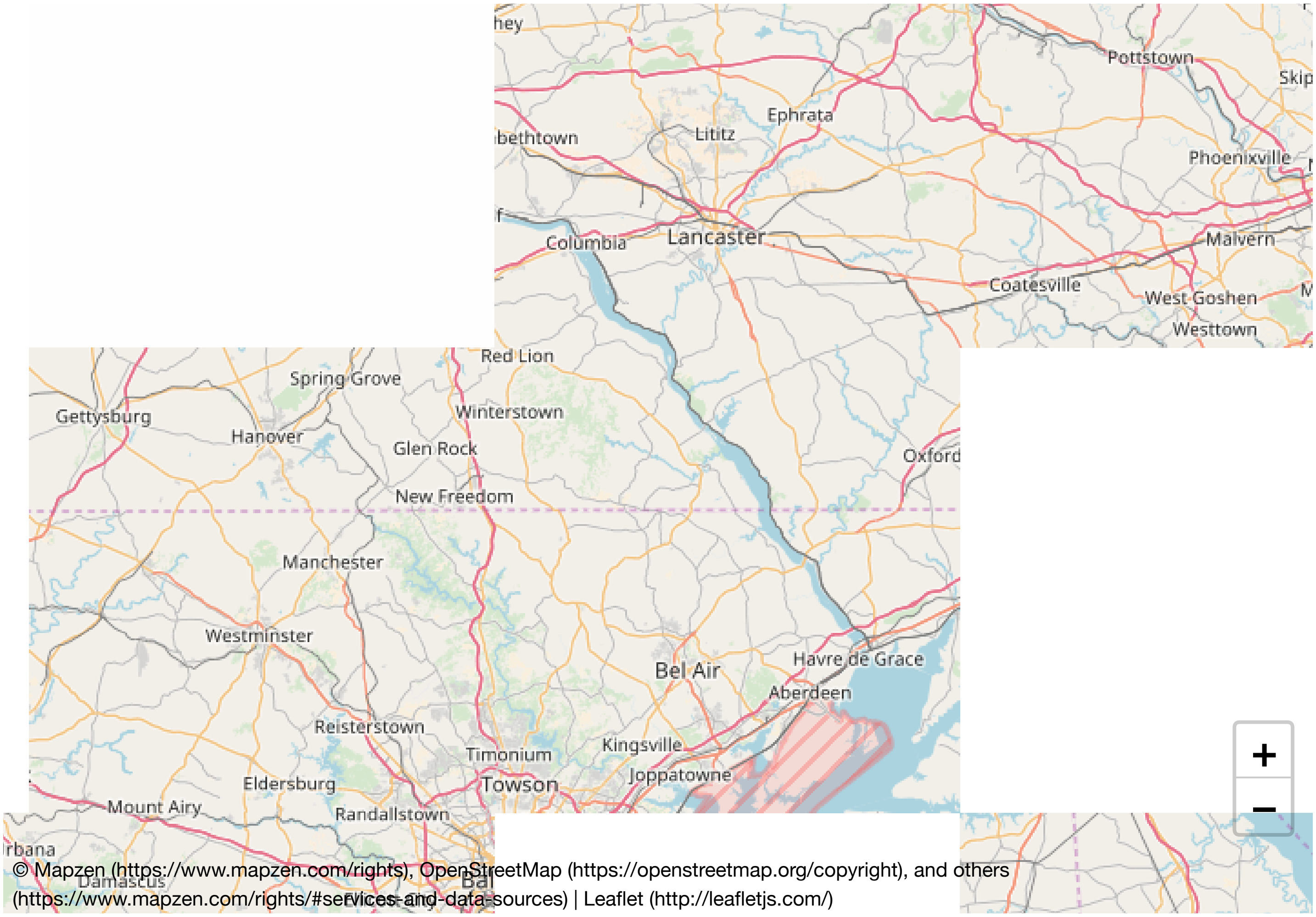
Real-time additions to transportation networks are often difficult to support, especially for hosted systems like Mapzen's Mobility services, which support world-wide transportation networks. However, creating mechanisms to remove or temporarily disable specific sections of roadway or transit lines are often possible. We have recently added 2 such features to our hosted Mapzen Mobility services.

The first transportation "editing" feature we have added is the ability to filter or limit the use of various public transit data within routes, time-distance matrices, and isochrones. These filters were discussed in detail **in a recent blog post (https://mapzen.com/blog/exclusion/)**

These filters allow inclusion of a limited list of transit operators, stops, or routes. They also allow exclusion of specific operators, stops, or routes. For example, one could exclude or remove a specific transit line (route) and view the impact via the isochrone service. This is supported within the Mobility Explorer, or a client or user could call the Mapzen Isochrone service and render the GeoJSON directly.

Another editing feature that was recently added is the ability to specify "avoid locations" within the route, time-distance matrix, or isochrone request. These avoid locations are mapped to the nearest road segment (or multiple road segments if the location is at an intersection) and these road segments are disallowed in the result. This can be useful to analyze the impact of a potential bridge closure. The following images show isochrones before and after avoiding the Norman Wood Bridge between Lancaster and York counties in Pennsylvania, which was closed in September 2015. Isochrones allow visualization of the impact to residents who needed to travel to from York county (west side of the Susquehanna River) into Lancaster county. Note that a large power plant is located just south of PA 372 near the bridge employs many people in both sides of the river. Commute times were dramatically increased for some locations (from under 15 minutes to over an hour).

Normal Travel Times


hey
Pottstown
Skip
Ephrata
Lititz
bethtown
Phoenixville
Columbia
Lancaster
Malvern
Coatesville
West Goshen
Westtown
Red Lion
Spring Grove
Gettysburg
Winterstown
Hanover
Glen Rock
Oxford
New Freedom
Manchester
Westminster
Bel Air
Havre de Grace
Aberdeen
Reisterstown
Kingsville
Timonium
Joppatowne
Eldersburg
Towson
Mount Airy
Randallstown
rbana
Damascus
Bal
+
−

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

Open this map full-screen (https://mapzen.com/tangram/view/?scene=https://mapzen.com/api/scenes/22/520/resources/default.yaml#9.8125/39.8815/-76.3885), and **check out the source in Tangram Play (https://mapzen.com/tangram/play/?scene=https://mapzen.com/api/scenes/22/520/resources/default.yaml#9.8125/39.8815/-76.3885)**!

This is what an isochrone API call looks like:

```
https://matrix.mapzen.com/isochrone?api_key=mapzen-xxxxxx&json={"locations":[{"lat":"43.0
```

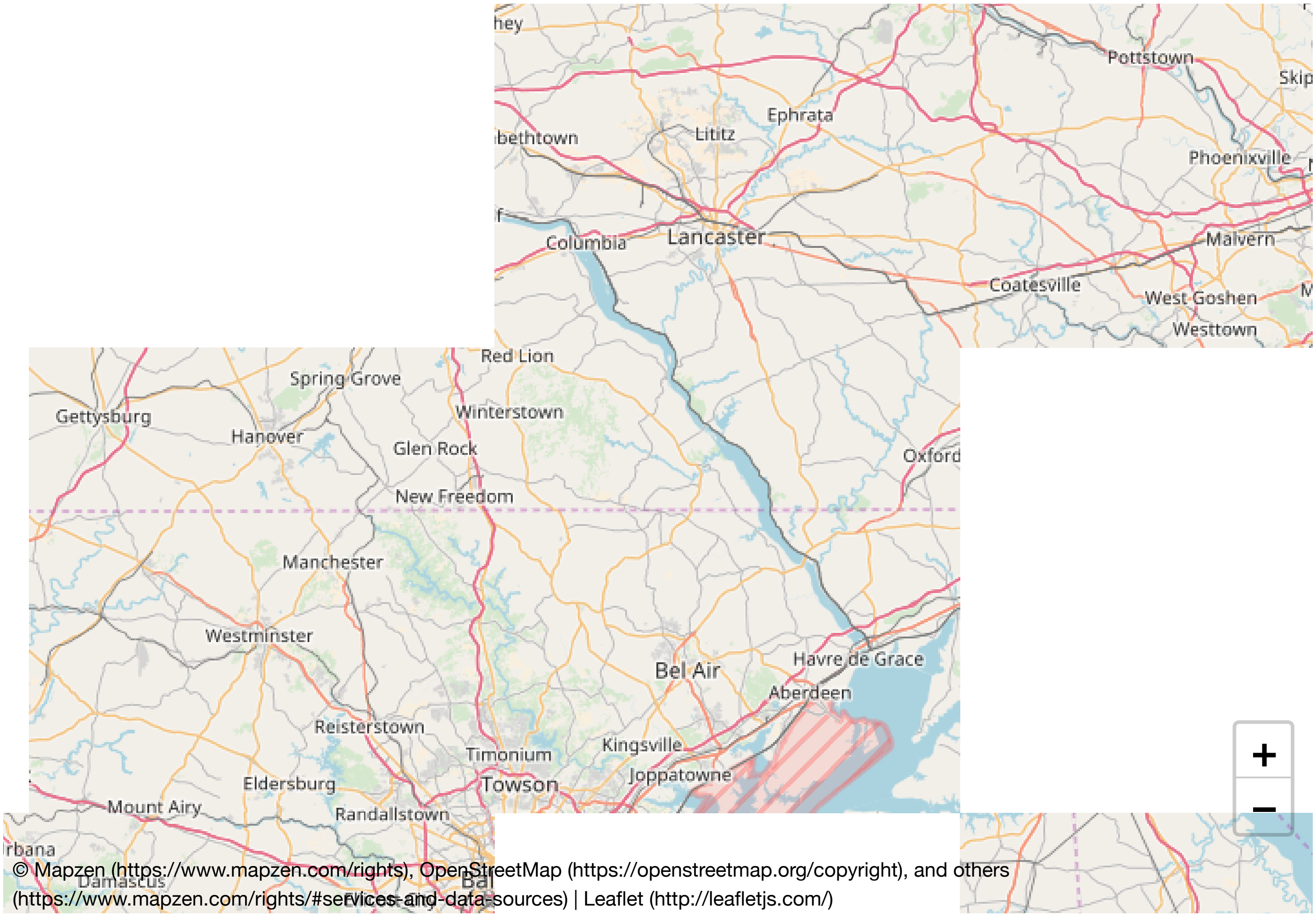
If you add your **API key (https://mapzen.com/developers)** in place of `mapzen-xxxxxx` , you can see the GeoJSON it returns. (We've also **cached the response (https://s3.amazonaws.com/mapzen-assets/images/post-isochrone/regular.geojson)** for your convenience.)

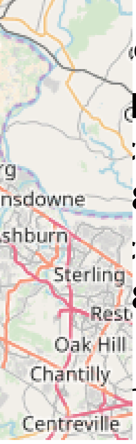
You can take this GeoJSON and draw it in many different mapping applications, as we have in Tangram below, or geojson.io, or Leaflet, or CARTO, or or QGIS...

To `exclude` the Norman Wood Bridge, we add the `avoid` parameters and the lat/lon of the bridge to the API call:

```
{"locations":[{"lat":39.795002,"lon":-76.362720}],"avoid_locations":
[{"lat":39.815447,"lon":-76.325397}],"costing":"auto"}
```

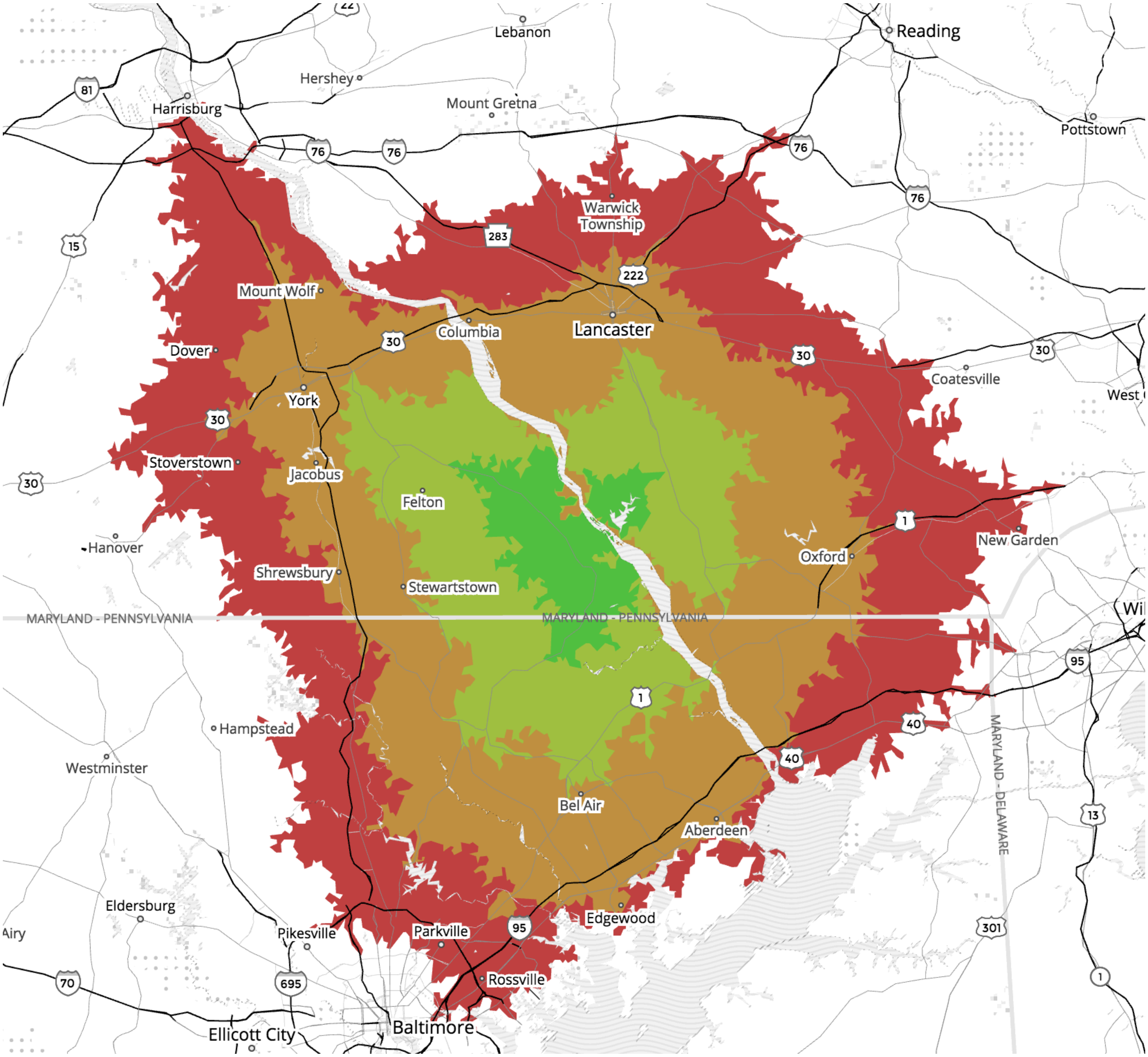
Travel Times with Norman Wood Bridge Disabled:


Pottstown
Ephrata
Lititz
Phoenixville
Columbia
Lancaster
Malvern
Coatesville
West Goshen
Westtown
Red Lion
Spring Grove
Gettysburg
Hanover
Winterstown
Glen Rock
Oxford
New Freedom
Manchester
Westminster
Bel Air
Havre de Grace
Aberdeen
Reisterstown
Timonium
Kingsville
Joppatowne
Towson
Eldersburg
Mount Airy
Randallstown
+
−
© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)


nsdowne
shburn
Sterling
Rest
Oak Hill
Chantilly
Centreville

**pen this map full-screen! (https://mapzen.com/tangram/view/?
:ene=https://mapzen.com/api/scenes/22/519/resources/default.yaml#9.8125/39.8815/-76
878)**, and **check out the source in Tangram Play (https://mapzen.com/tangram/play/?
:ene=https://mapzen.com/api/scenes/22/519/resources/default.yaml#9.8125/39.8815/-76
878)**!

ıd naturally, here's a handy comparison GIF:


Reading
Lebanon
Hershey
Harrisburg
Mount Gretna
Pottstown
Warwick Township
Mount Wolf
Columbia
Lancaster
Dover
York
Coatesville
West
Stoverstown
Jacobus
Felton
Hanover
Shrewsbury
Stewartstown
Oxford
New Garden
MARYLAND - PENNSYLVANIA
Hampstead
Westminster
Bel Air
Aberdeen
MARYLAND - DELAWARE
Eldersburg
Airy
Pikesville
Parkville
Edgewood
Rossville
Ellicott City
Baltimore
81
76
283
222
15
30
1
95
40
13
301
70
695

Isochrones along with inclusion and exclusion are a powerful tool as means of supporting transportation analysis applications. Learn more about them, or **drop us a line (mailto:hello@mapzen.com?subject=isochrones)**!

- **Introducting Mapzen Mobility Explorer (https://mapzen.com/blog/introducing-mobility-explorer/)** (blog post)
- **Mapzen Mobility Explorer (https://mapzen.com/mobility/explorer/)**
- **Transit Inclusion and Exclusion (https://mapzen.com/blog/exclusion/)** (blog post)
- **Introducing Isochrones (https://mapzen.com/blog/introducing-isochrone-service/)** (blog post)
- **Mapzen Isochrone Documentation (https://mapzen.com/documentation/mobility/explorer/isochrones/)** (docs)

· 19 April 2017 ·

Greg Knisely

Greg is a data munger and software engineer for Mapzen's routing software.

David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

Meghan Hade

Meghan is a software engineer with a background in urban planning on Mapzen's Mobility team. She works in javascript and plays in calligraphy, block printing, and finding ways to stash n+1 bikes in a San Francisco apartment.

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

Updating Who's On First Neighbourhoods - Part II

whosonfirst (/tag/whosonfirst) data (/tag/data)

In a **previous blog post (https://mapzen.com/blog/sf-neighbourhood-updates/)**, we outlined Mapzen's efforts to update neighbourhood records in Who's On First. Since that blog post was published, we've been busy updating our records in several major metropolitan areas like Washington D.C., Helsinki, Amsterdam, Chicago and New Orleans, which are all now available in Mapzen's **basemap styles (https://mapzen.com/products/maps/)**.

We've not only updated our records, but we've also developed a streamlined approach to importing **new sources (https://github.com/whosonfirst/whosonfirst-sources/tree/master/sources)** and adding **new properties (https://github.com/whosonfirst/whosonfirst-properties/tree/master/aliases)** to existing records. In addition to neighbourhoods, Who's On First has gained new and updated microhood, macrohood, borough, and locality records. For more information on how we define our placetypes, see our handy **placetypes (https://github.com/whosonfirst/whosonfirst-placetypes)** repository.

How do we update records?

We want to make it straightforward for anyone to contribute, so all of the standard tools Mapzen uses are open-source and freely available for download. The major software tool we use to manually enhancing our Who's On First data is the **QGIS (http://www.qgis.org/en/site/)** open-source geographic information system.

Our Workflow

Our workflow revolves around finding authoritative geospatial neighbourhood information and integrating that into our current inventory of records. That means importing new neighbourhoods from open sources and updating current neighbourhood records, often with sourced geometries and attributes. In most situations we reviewed and modified local authoritative sources, but in certain situations where open data was not available, we hand-crafted new neighbourhood shapes.

Additional Resources

On the surface, the process to update these neighbourhoods is simple, but can get more challenging. Neighbourhoods are usually represented as polygons, but sometimes they are points. We have to consider naming conventions and languages across different sources, as well as making sure neighbourhoods don't extend outside their locality.

For a more-detailed technical rundown on our process, our updated **document on WOF neighbourhoods (https://github.com/whosonfirst/whosonfirst-cookbook/blob/master/issue_workflows/sf_neighbourhood_updates_pt_1.md)** is a good place to start.

Where do we find sources?

In our mission to improve Who's On First records, we often turn to municipal or other authoritative open data portals to furnish the framework for our neighbourhood data. Local governments often provide high-quality maps and data of their neighbourhoods in order to aid in planning or community development efforts, which we can then use to provide more reliable information in Who's On First.

However, not all data needs to come from government sources. Just as regularly, we use detailed, open neighbourhood data from private entities. Regardless of the source, all data is distributed with a permissive license, often coming in the form of a **CC0 (https://creativecommons.org/share-your-work/public-domain/cc0/)** or a **CC-BY (https://creativecommons.org/licenses/by/4.0/)** license.

How can you view sources?

All of the sources we use are listed in our detailed **sources page (https://github.com/whosonfirst/whosonfirst-sources/tree/master/sources#sources)**. If you ever wonder where a particular bit of information or property comes from, that page will direct you to its source. This list is updated whenever a new, open-license source is incorporated into Who's On First. To search an individual source, simply look through our list for the appropriate reference code ("mz", "wof", etc.) applicable to the data in question.

Where have neighbourhood records been updated?

We've been busy updating neighbourhood (and associated placetype) shapes; as of this blog post, we have updated administrative features in 35 localities! A list of those places, alphabetically:

- **Amsterdam (https://whosonfirst.mapzen.com/spelunker/download/101751893/)**
- **Atlanta (https://whosonfirst.mapzen.com/spelunker/download/85936429/)**
- **Baltimore (https://whosonfirst.mapzen.com/spelunker/download/102081589/)**
- **Berlin (https://whosonfirst.mapzen.com/spelunker/download/101748799/)**
- **Boston (https://whosonfirst.mapzen.com/spelunker/download/85950361/)** (including **Cambridge (https://whosonfirst.mapzen.com/spelunker/download/85950329/)** and **Somerville (https://whosonfirst.mapzen.com/spelunker/download/85950339/)**)
- **Chicago (https://whosonfirst.mapzen.com/spelunker/download/85940195/)**
- **Denver (https://whosonfirst.mapzen.com/spelunker/download/85928879/)**
- Greater Finland (includes **Vantaa (https://whosonfirst.mapzen.com/spelunker/download/101748419/)**, **Turku (https://whosonfirst.mapzen.com/spelunker/download/101748421/)**, **Tampere (https://whosonfirst.mapzen.com/spelunker/download/101748431/)**, **Oulu (https://whosonfirst.mapzen.com/spelunker/download/101748443/)**, **Kuopio (https://whosonfirst.mapzen.com/spelunker/download/101748439/)**, **Kauniainen (https://whosonfirst.mapzen.com/spelunker/download/101753897/)**, and **Espoo (https://whosonfirst.mapzen.com/spelunker/download/101748415/)**)
- **Helsinki (https://whosonfirst.mapzen.com/spelunker/download/101748417/)**
- **Los Angeles (https://whosonfirst.mapzen.com/spelunker/download/85923517/)**
- **Minneapolis (https://whosonfirst.mapzen.com/spelunker/download/85969169/)** & **St. Paul (https://whosonfirst.mapzen.com/spelunker/download/85953191/)**
- **New Orleans (https://whosonfirst.mapzen.com/spelunker/download/85948111/)**
- **New York City (https://whosonfirst.mapzen.com/spelunker/download/85977539/)**
- **Oakland (https://whosonfirst.mapzen.com/spelunker/download/85921881/)**
- **Philadelphia (https://whosonfirst.mapzen.com/spelunker/download/102081353/)**
- **Portland (https://whosonfirst.mapzen.com/spelunker/download/101715829/)**
- **Rotterdam (https://whosonfirst.mapzen.com/spelunker/download/101751843/)**
- **San Diego (https://whosonfirst.mapzen.com/spelunker/download/85922227/)**
- **San Francisco (https://whosonfirst.mapzen.com/spelunker/download/85922583/)**
- **San Jose (https://whosonfirst.mapzen.com/spelunker/download/85922347/)**
- **Seattle (https://whosonfirst.mapzen.com/spelunker/download/101730401/)**

- **The Hague (https://whosonfirst.mapzen.com/spelunker/download/101752763/)**
- **Utrecht (https://whosonfirst.mapzen.com/spelunker/download/101751827/)**
- **Washington, D.C. (https://whosonfirst.mapzen.com/spelunker/download/85931779/)** (including **Arlington (https://whosonfirst.mapzen.com/spelunker/download/101729469/)** and **Alexandria (https://whosonfirst.mapzen.com/spelunker/download/102080641/)**)

Nice list, but how do the new shapes compare?

We can talk all we want about new neighbourhood shapes, but it's probably best to show these new shapes in **Tangram (https://mapzen.com/tangram/play/)**. Below, we have two localites to compare, pre-update and post-update: Denver and Amsterdam.

Note that the original shapes have a more organic, popcorn-looking shape, while the new shapes are more regularly aligned along arterial roads and other major landmarks.

Denver, Colorado, USA:

- **Original neighbourhood** features are shown **up to** zoom 11 (default map zoom) **in orange**.
- **New macrohood** features are shown from zoom 11 to zoom 12 **in blue**.
- **New and updated neighbourhood** features are shown from zoom 12 to zoom 13 **in green**.
- **New microhood** features are shown **past** zoom 13 **in purple**.

Click to view Denver in full screen ➶ (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/neighbourhood-updates-II/denver.yaml#10.5/39.7609/-104.8612)

Amsterdam, Netherlands:

- **Original neighbourhood** features are shown **up to** zoom 11 (default map zoom) **in orange**.
- **New borough** features are shown from zoom 11 to zoom 12 **in blue**.
- **New and updated neighbourhood** features are shown from zoom 12 to zoom 13 **in green**.
- **New microhood** features are shown **past** zoom 13 **in purple**.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Click to view Amsterdam in full screen ↗ (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/neighbourhood-updates-II/amsterdam.yaml#10.5/52.3553/4.8892)

In conclusion…

Neighbourhood updates are not over yet; we plan on continuing this work and have many more localities in the pipeline. Keep an eye on our **Who's On First data repository (https://github.com/whosonfirst-data/whosonfirst-data)** for continued updates!

If you have a specific request or suggestion, please do not hesitate to **contact us (mailto:hello@mapzen.com?subject=%22Who's%20On%20First%20neighbourhoods%22)**! Thanks for reading!

Photo Credit: ***midnightboheme, Pixabay (https://pixabay.com/en/new-orleans-neighborhoods-map-river-1228029/)***

· 20 April 2017 ·

Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.

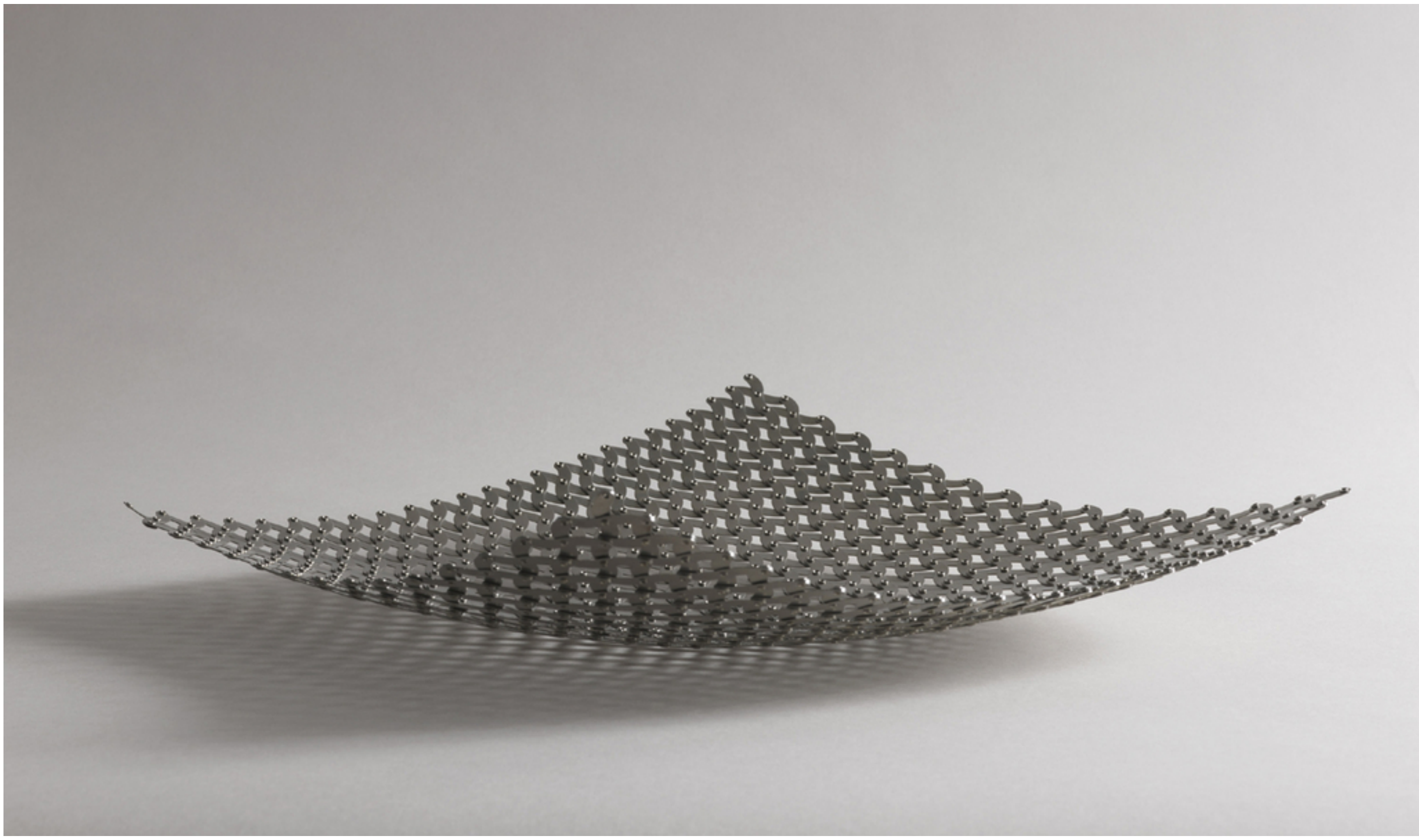
Zach Singleton

Zach is a GIS enthusiast who happens to have an unhealthy relationship with all things spicy.

© 2017 Mapzen

Introducing Mapzen Flex

vector-tiles (/tag/vector-tiles) routing (/tag/routing) search (/tag/search) flex (/tag/flex) mobility (/tag/mobility)



Almost two years ago we launched a hosted **vector tile service (https://mapzen.com/blog/look-upon-our-squares-of-math-in-three-dimensions)**. We followed it quickly with a **routing engine (https://mapzen.com/blog/introducing-valhalla)** and a **geocoder (https://mapzen.com/blog/the-world-is-yours-announcing-mapzen-search)**. Since then, we've signed up tens of thousands of developers onto the Mapzen platform, **improved (https://mapzen.com/blog/v1-vector-tile-service/) our (https://mapzen.com/blog/valhalla-2.0/) products (https://mapzen.com/blog/the-next-chapter-of-search/)**, and worked hard to deliver our services at scale.

Our most frequent request has been how to get more access to Mapzen's hosted services. Since the beginning, we've rate limited our free platform to ensure access for all our developers, but many have needed more.

Today we're excited to introduce **Mapzen Flex (https://mapzen.com/pricing)**, the new way we will offer our hosted services.

Just like before, every service will have a free tier. If you don't go above that, not much will change. The difference? Now you can pay to exceed the free request cap, starting with vector tiles, search, autocomplete and turn-by-turn navigation services. (We'll be adding the rest of our services to Mapzen Flex in the coming months.)

In order to be sustainable and fair, we wanted to find a pricing plan in line with our greater goals of openness and accessibility. By charging a **fixed, transparent price (https://mapzen.com/pricing)** across all our products, we hope to improve how developers buy mapping services by giving them confidence and clarity. Instead of bundles or packages or tiers, you just pay a few cents per thousand requests for what you use above the free monthly limit.

Need more? No problem. Need less? We won't charge you extra. And no surprises! Set budget limits and we'll let you know when you're getting close.

We also think it's important to treat all of our customers equally. Whether your app is on mobile or desktop, you'll pay the same rate. Whether your app is free or paid, the same rate applies.

We'll be rolling out the new rate limits on June 1st, but if you need higher rate limits now, **contact us (emailto:pricing@mapzen.com)** to get going on Mapzen Flex.

Thanks for being with us for the past few years! We're excited about Mapzen Flex; it will allow us to continue to grow at scale, help our products become even better, and meet our users' needs for more access to our services.

Preview image ***via Cooper Hewitt (https://collection.cooperhewitt.org/objects/18714299/)****: up-curved form composed of joined steel segments that flex to expand or contract into square or rectangle, designed by Fernando Campana and Humberto Campana*

· 25 April 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

The Who's On First API Explorer

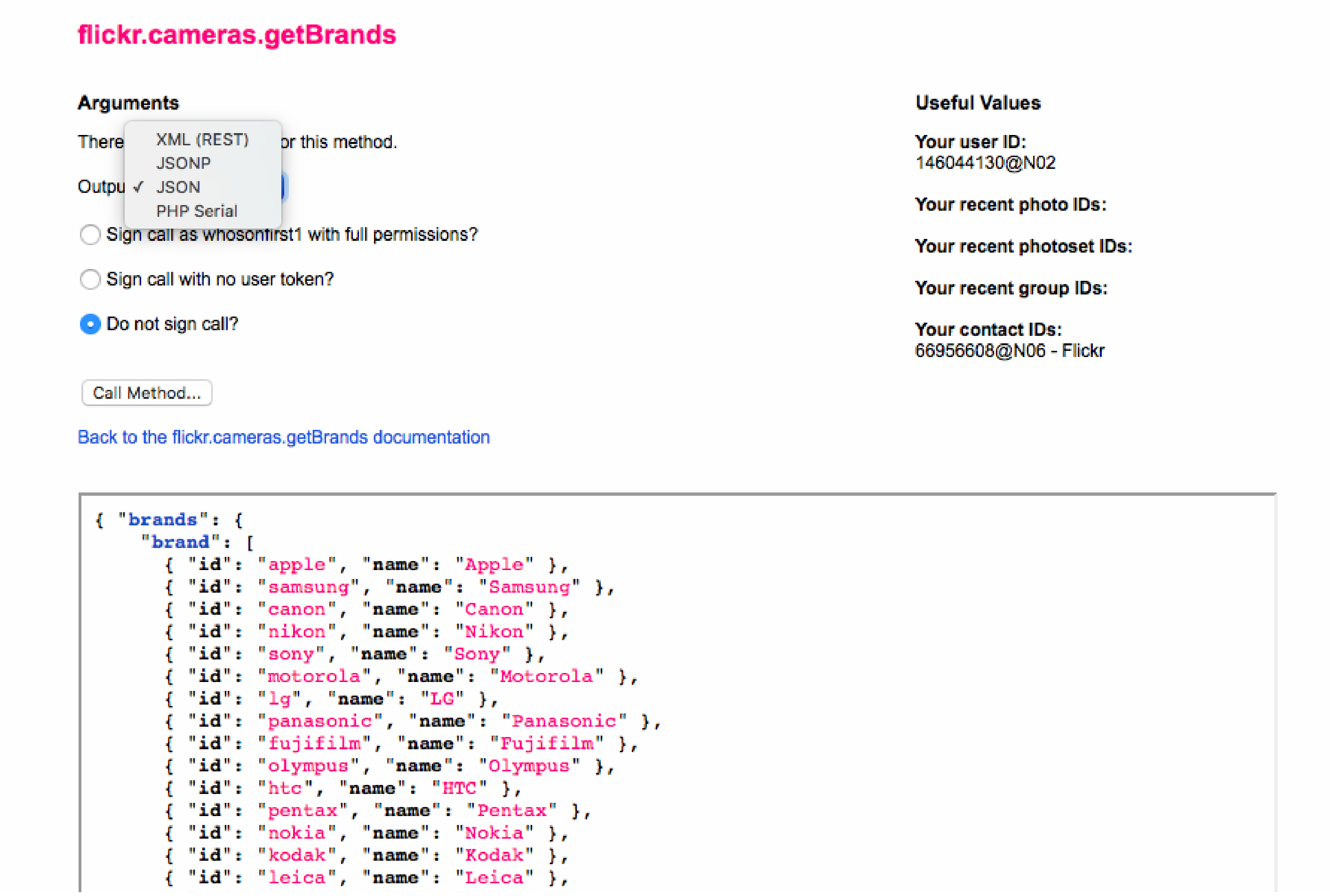
whosonfirst (/tag/whosonfirst)

(https://collection.cooperhewitt.org/objects/18419753/)

A few weeks ago **Lou built an Electron application (https://mapzen.com/blog/tangram-work-developer-commentary)**. For April Fools' Day he made a version of **Tangram Play (https://mapzen.com/blog/introducing-tangram-work)** in the style of a 90's era Windows application and published it as a living breathing desktop application rather than **the web application (https://mapzen.com/tangram/play/)** we've come to know and love. In addition he was able to publish MacOS and Linux native applications, all from the same code base. The same code base that is ultimately just HTML and CSS and Javascript (aka "a web application"). An **increasing number of desktop applications (https://electron.atom.io/apps)** are being developed this way and it's what makes Electron pretty exciting.

Shortly after Lou's blog post we pushed the **Who's On First API (https://mapzen.com/blog/whosonfirst-api)** out the door. One of the features of the API is that it exposes a suite of `api.spec.*` methods **that allow you to introspect the API itself (https://mapzen.com/documentation/wof/methods/#apispec)**. You can fetch all of the **methods (https://mapzen.com/documentation/wof/methods/#api.spec.methods)** (and their parameters and errors), all of the **common errors (https://mapzen.com/documentation/wof/methods/#api.spec.errors)** and all of the supported **output formats (https://mapzen.com/documentation/wof/methods/#api.spec.formats)**. These methods owe a debt of inspiration to the nice people at **Linode (http://linode.com)** who started doing something similar **way back in 2012 (http://blog.linode.com/2012/04/04/api_spec/)** and from which the `api.spec` naming convention is derived.

The Who's On First API has always enjoyed a internal web-based "explorer" tool. API explorers are pretty common these days but if you're not sure they are **have a look at the Flickr API Explorer (https://www.flickr.com/services/api/)** which is pretty much the ur-Explorer that started them all. Go ahead, we'll wait…



flickr.cameras.getBrands

Arguments

There r this method.

Outpu JSON

○ Sign call as whosonfirst1 with full permissions?

○ Sign call with no user token?

● Do not sign call?

Back to the flickr.cameras.getBrands documentation

Useful Values

Your user ID:
146044130@N02

Your recent photo IDs:

Your recent photoset IDs:

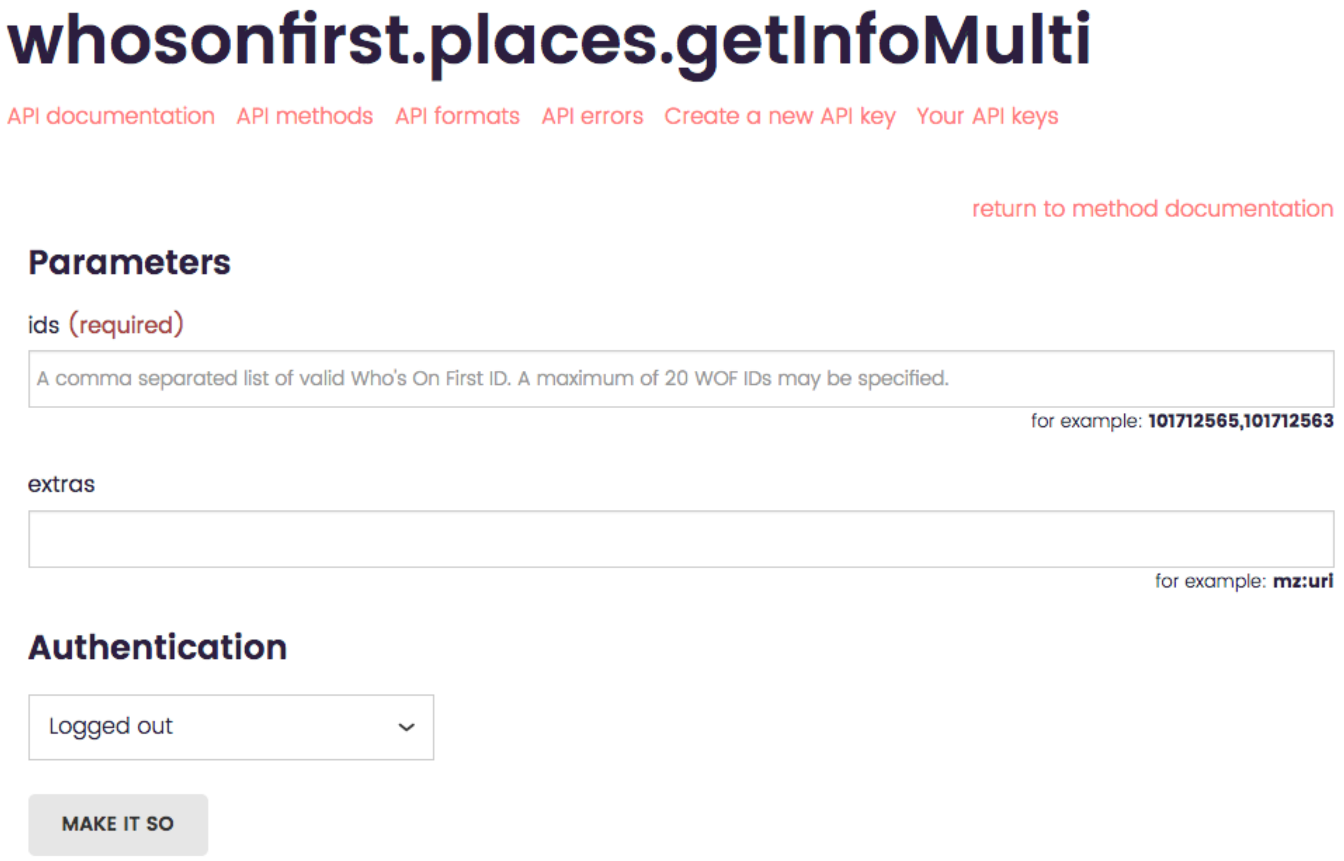
Your recent group IDs:

Your contact IDs:
66956608@N06 - Flickr

```
{ "brands": {
    "brand": [
      { "id": "apple", "name": "Apple" },
      { "id": "samsung", "name": "Samsung" },
      { "id": "canon", "name": "Canon" },
      { "id": "nikon", "name": "Nikon" },
      { "id": "sony", "name": "Sony" },
      { "id": "motorola", "name": "Motorola" },
      { "id": "lg", "name": "LG" },
      { "id": "panasonic", "name": "Panasonic" },
      { "id": "fujifilm", "name": "Fujifilm" },
      { "id": "olympus", "name": "Olympus" },
      { "id": "htc", "name": "HTC" },
      { "id": "pentax", "name": "Pentax" },
      { "id": "nokia", "name": "Nokia" },
      { "id": "kodak", "name": "Kodak" },
      { "id": "leica", "name": "Leica" },
```

*Also, Cal Henderson's 2007 talk **"Flickr and APIs: A Love Story" (http://www.iamcal.com/talks/open_hack_day.pps)** which is a 22MB PowerPoint presentation because… it was 2007, I guess. Anyway, it it a good talk.*

We use our own API Explorer extensively for debugging and generally poking around. It is often the fastest way to answer a question about the data. I find it difficult to imagine developing or working with APIs *without* an API explorer now and that makes the internal-iness of the WOF API Explorer less than awesome. For a variety of reasons it won't be made public any time soon which is sort of like pouring salt in your own wound.



whosonfirst.places.getInfoMulti

Parameters

ids (required)

A comma separated list of valid Who's On First ID. A maximum of 20 WOF IDs may be specified.

for example: **101712565,101712563**

extras

for example: **mz:uri**

Authentication

Logged out

By now, some of you might have figured out where this is going: If the WOF API has a full suite of methods for inspecting the API and Electron makes it easy to develop custom applications in HTML and Javascript then why not build… **a standalone Who's On First API Explorer (https://github.com/whosonfirst/electron-whosonfirst-api-explorer)**. So we did!

After I showed an early version to a friend he asked *"Why an Electron application?"* for an API explorer. Aside from the reasons outlined above I offered the following rationale:

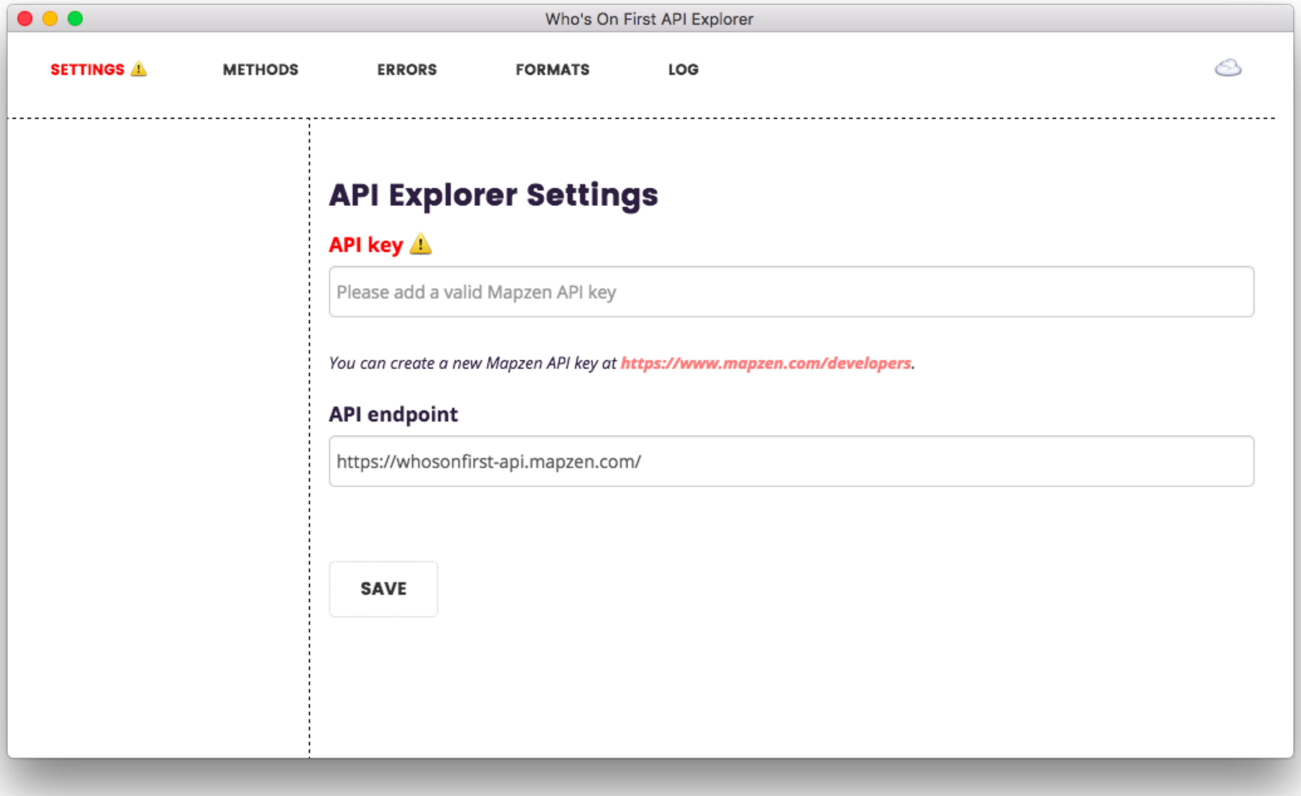
1. Curiousity. Does the Who's On First API lend itself to an Electron application? (It does!)

2. If you set aside some boring details about code-signing and paying Apple or Microsoft for the privilege, there is the possibility of generating native platform specific binaries for Electron applications. That means never having to say *"Just install Node.js…"* to someone. You can replace **Node.js (https://nodejs.org/)** with any number of technologies and the dynamic is the same. Tools that don't require a laundry list of extra steps to get started with are important. Not everyone is interested in the technology details because they are busy being awesome in other endeavours.
3. It was (is) a good way to put the `api.spec` methods through their paces. We've been able to identify a bunch of places where the API doesn't return enough information to usefully build an explorer-style interface. Many have been fixed already and the rest (notably information about output formats) are slated to be improved shortly.
4. It allows us offer a public version of the API explorer.
5. It is a good way to figure out which out **the Who's On First Javascript libraries (https://github.com/whosonfirst/js-mapzen-whosonfirst)** should be rejiggered to hold hands with "modern Javascript" and Node.js – things like the **mapzen.whosonfirst.api.js (https://github.com/whosonfirst/electron-whosonfirst-api-explorer/blob/master/mapzen.whosonfirst.api.js)** library which we'd like to bless as officially supported alongside the **Go (https://github.com/whosonfirst/go-whosonfirst-api)** and **Python (https://github.com/whosonfirst/py-mapzen-whosonfirst-api)** libraries.
6. It works offline because initial calls to the `api.spec.*` methods are cached locally.

I like to think the WOF API Explorer is another illustration of the idea that **"Mapzen should always be Consumer Zero (of Mapzen services)" (https://mapzen.com/blog/iamhere)**. The WOF API Explorer is just one of many possible **targeted and focused applications (https://mapzen.com/blog/bundler)** that we might build with tools like Electron and **our own services (https://www.mapzen.com/documentation)**.

The **WOF API Explorer is available for anyone to try (https://github.com/whosonfirst/electron-whosonfirst-api-explorer)**. As of this writing, and despite what I said above, you will still need to *"Just install Node.js"* to get it to work. Hopefully the time when we dig in to the details of building and signing native applications will come soon. Until then there is **a little bit (possibly a lot, depending on your skill and comfort level) of preamble (https://github.com/whosonfirst/electron-whosonfirst-api-explorer#install)** you'll need to do to get the WOF API Explorer up and running.

In the interim, here are a bunch of screenshots to hopefully make that extra work seem worth doing:


Who's On First API Explorer
SETTINGS ⚠
METHODS
ERRORS
FORMATS
LOG
SAVE

API Explorer Settings

API key ⚠

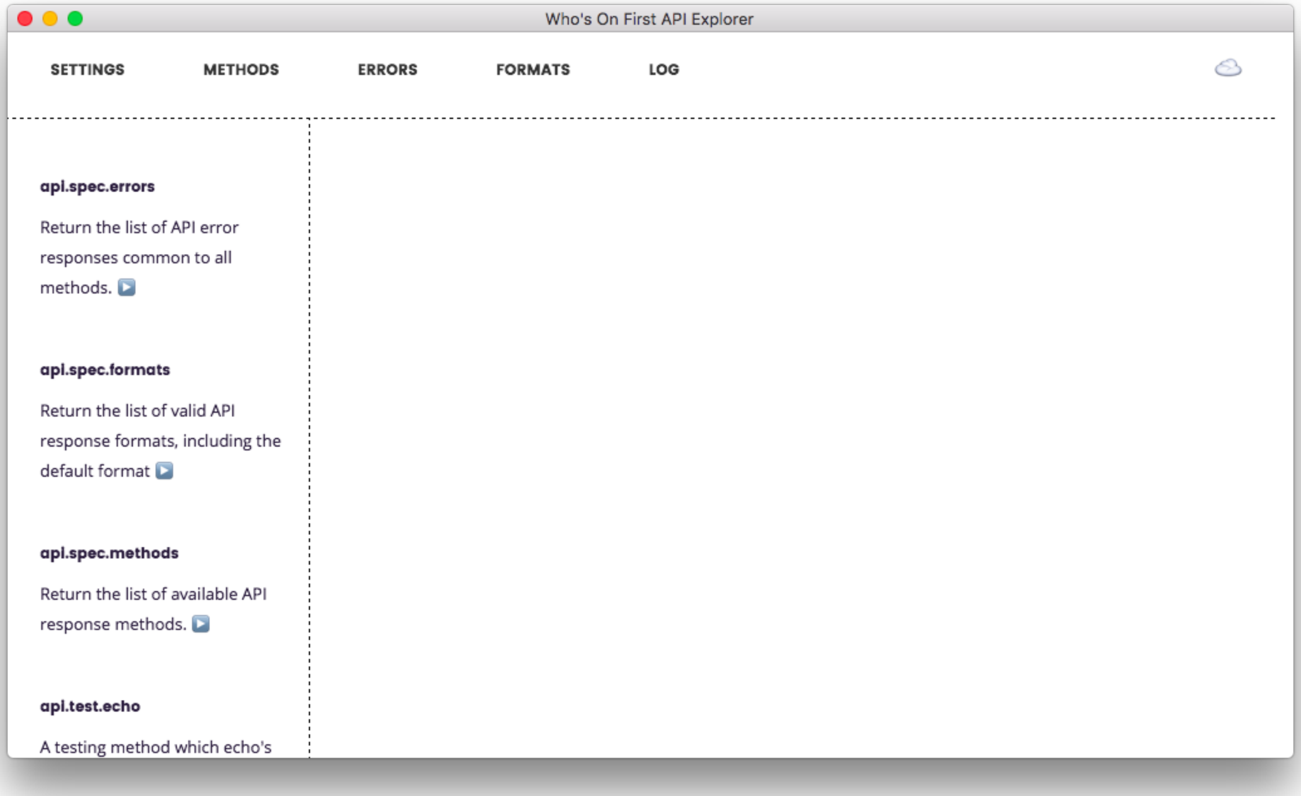
Please add a valid Mapzen API key

You can create a new Mapzen API key at https://www.mapzen.com/developers.

API endpoint

https://whosonfirst-api.mapzen.com/

Here's what happens when you start the API Explorer for the first time: ***You are asked for a Mapzen API key! (https://mapzen.com/developers)***



api.spec.errors

Return the list of API error responses common to all methods. ▶

api.spec.formats

Return the list of valid API response formats, including the default format ▶

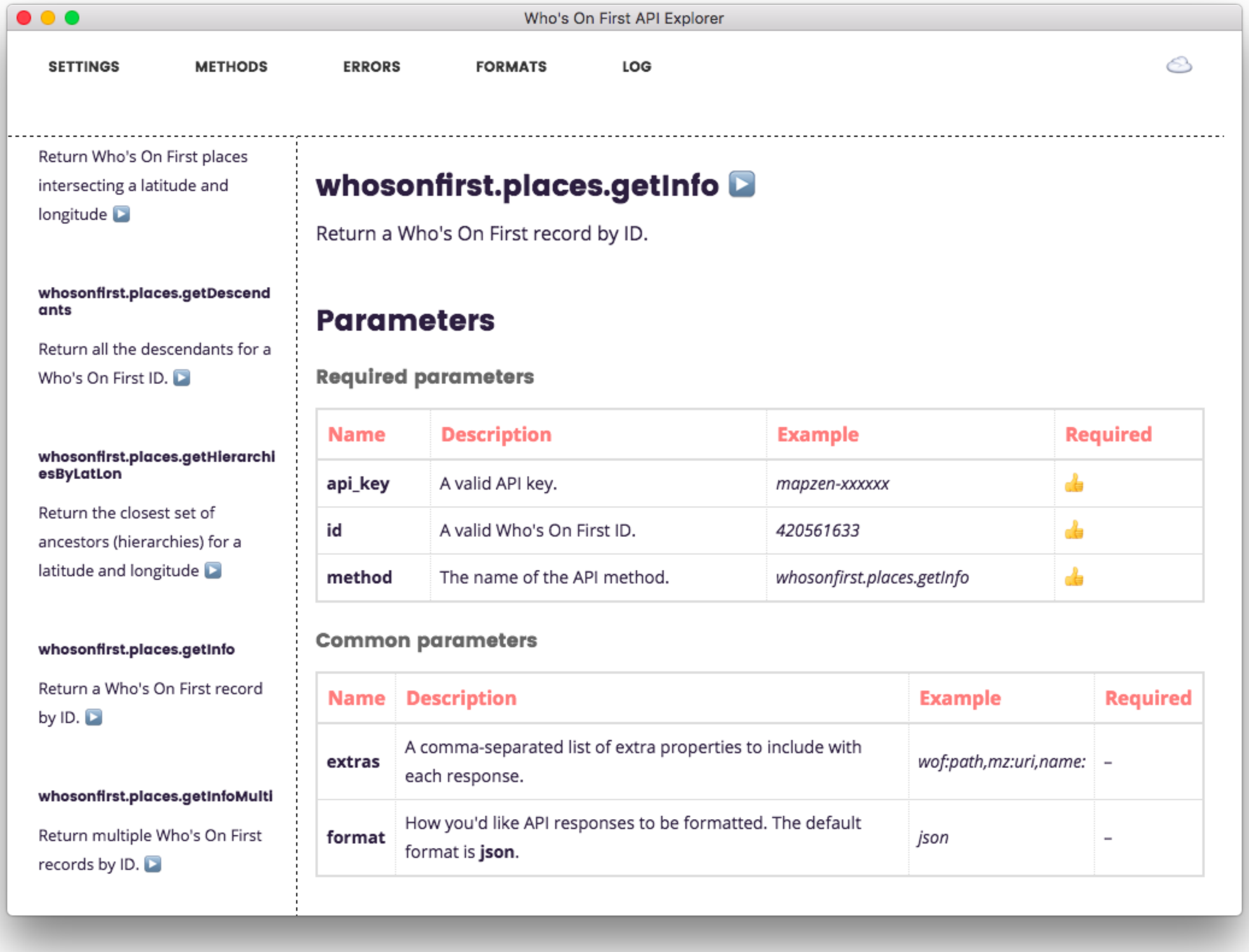
api.spec.methods

Return the list of available API response methods. ▶

api.test.echo

A testing method which echo's

List views appear in the left-hand sidebar. Here's a list of all the API methods. API methods, errors and formats are fetched from the API when the Explorer starts up and there is a handy "refresh" button at the bottom of each list view update things while the application is running.



Return Who's On First places intersecting a latitude and longitude

whosonfirst.places.getDescendants

Return all the descendants for a Who's On First ID.

whosonfirst.places.getHierarchiesByLatLon

Return the closest set of ancestors (hierarchies) for a latitude and longitude

whosonfirst.places.getInfo

Return a Who's On First record by ID.

whosonfirst.places.getInfoMulti

Return multiple Who's On First records by ID.

whosonfirst.places.getInfo

Return a Who's On First record by ID.

Parameters

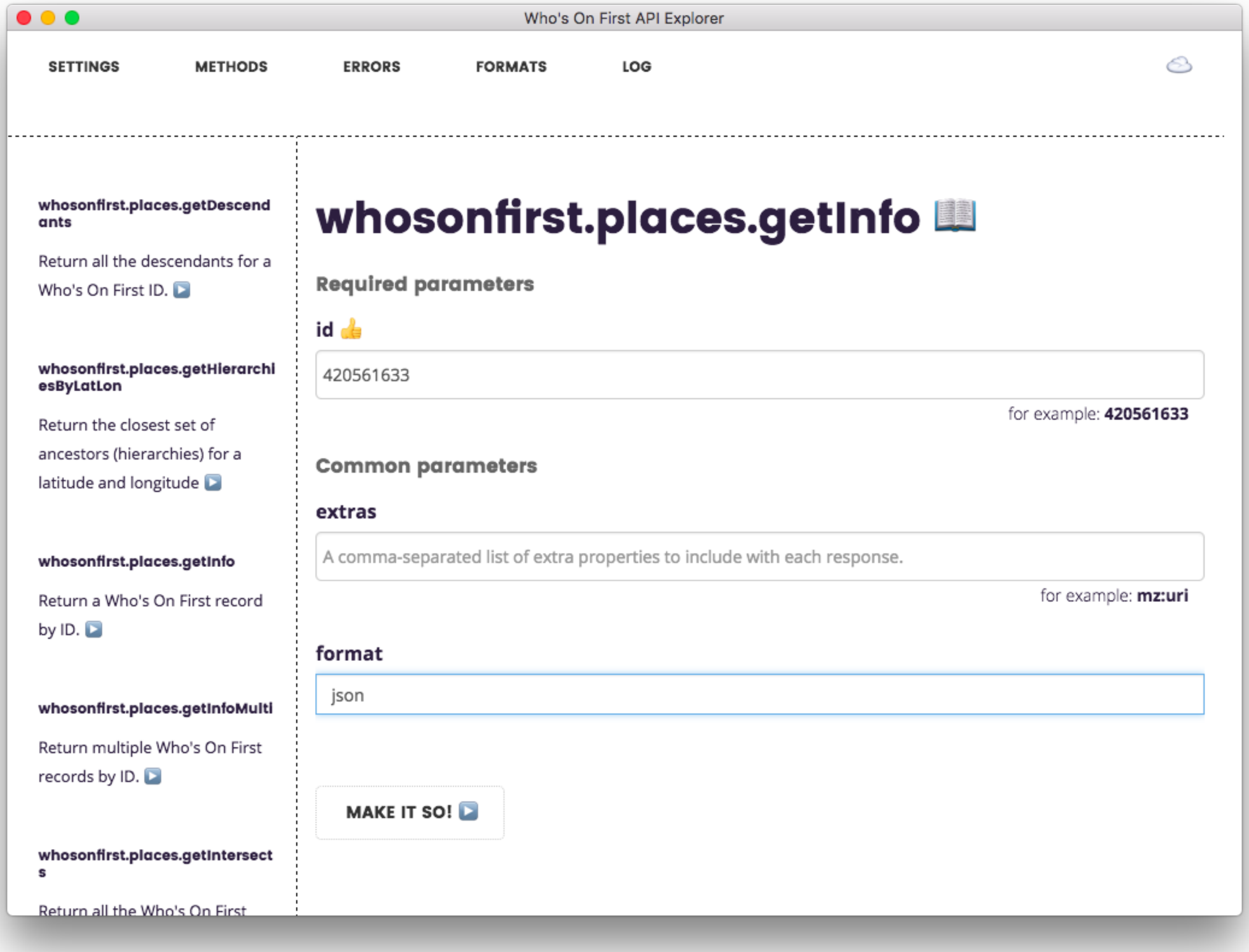
Required parameters

Name	Description	Example	Required
api_key	A valid API key.	*mapzen-xxxxxx*	👍
id	A valid Who's On First ID.	*420561633*	👍
method	The name of the API method.	*whosonfirst.places.getInfo*	👍

Common parameters

Name	Description	Example	Required
extras	A comma-separated list of extra properties to include with each response.	*wof:path,mz:uri,name:*	–
format	How you'd like API responses to be formatted. The default format is **json**.	*json*	–

*Here's an example of the documentation for the **whosonfirst.places.getInfo (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getInfo)** API method, which isn't that different from what you'd see on the **Mapzen website (https://mapzen.com/documentation/wof/)**.*



whosonfirst.places.getDescendants

Return all the descendants for a Who's On First ID. ▶

whosonfirst.places.getHierarchiesByLatLon

Return the closest set of ancestors (hierarchies) for a latitude and longitude ▶

whosonfirst.places.getInfo

Return a Who's On First record by ID. ▶

whosonfirst.places.getInfoMulti

Return multiple Who's On First records by ID. ▶

whosonfirst.places.getIntersects

Return all the Who's On First

whosonfirst.places.getInfo 📖

Required parameters

id 👍

420561633

for example: **420561633**

Common parameters

extras

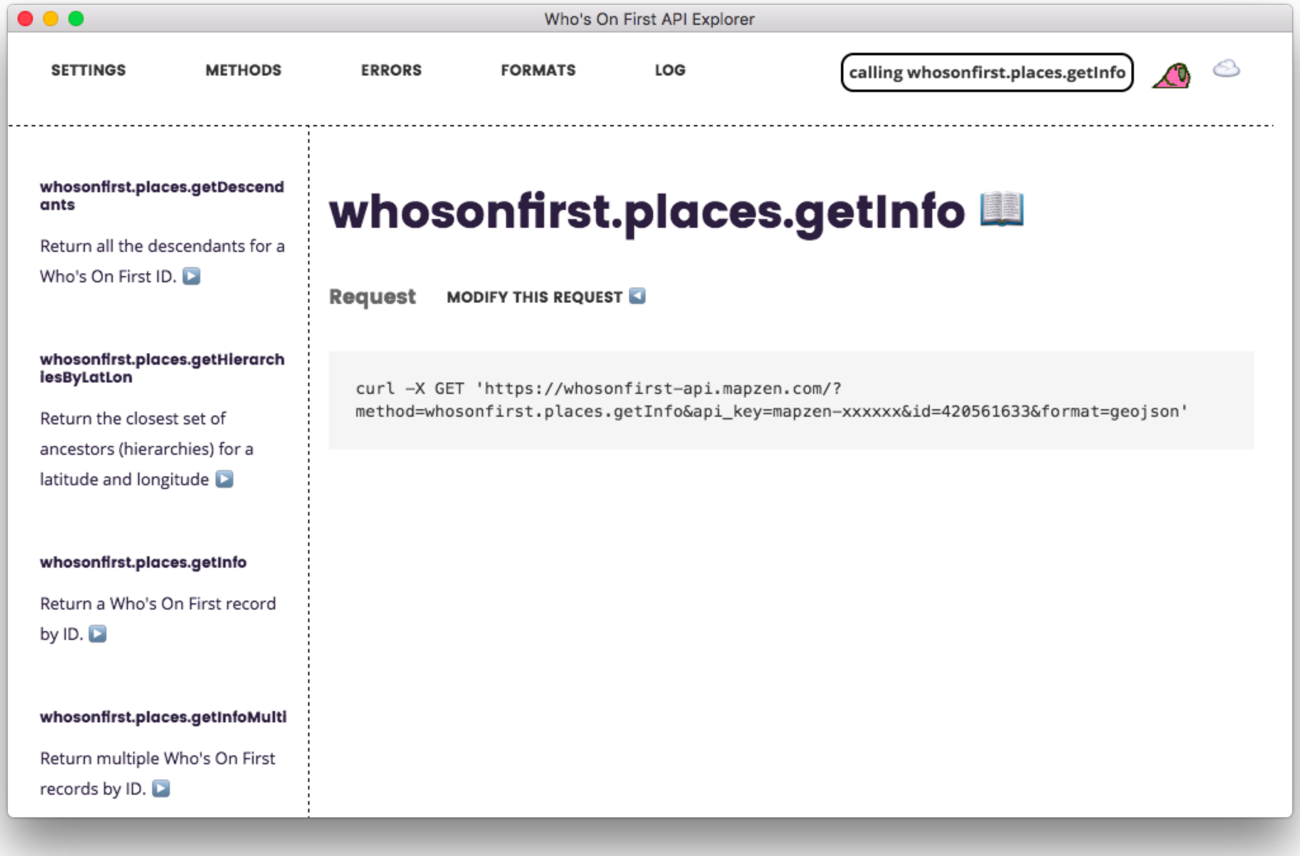
A comma-separated list of extra properties to include with each response.

for example: **mz:uri**

format

json

From any documentation page you can click through to a corresponding "explore" panel for that method. You can update the parameters that will be sent to the API and then click the handy `MAKE IT SO!` *button to start the API request.*


Who's On First API Explorer
SETTINGS
METHODS
ERRORS
FORMATS
LOG
calling whosonfirst.places.getInfo

whosonfirst.places.getDescendants

Return all the descendants for a Who's On First ID.

whosonfirst.places.getHierarchiesByLatLon

Return the closest set of ancestors (hierarchies) for a latitude and longitude

whosonfirst.places.getInfo

Return a Who's On First record by ID.

whosonfirst.places.getInfoMulti

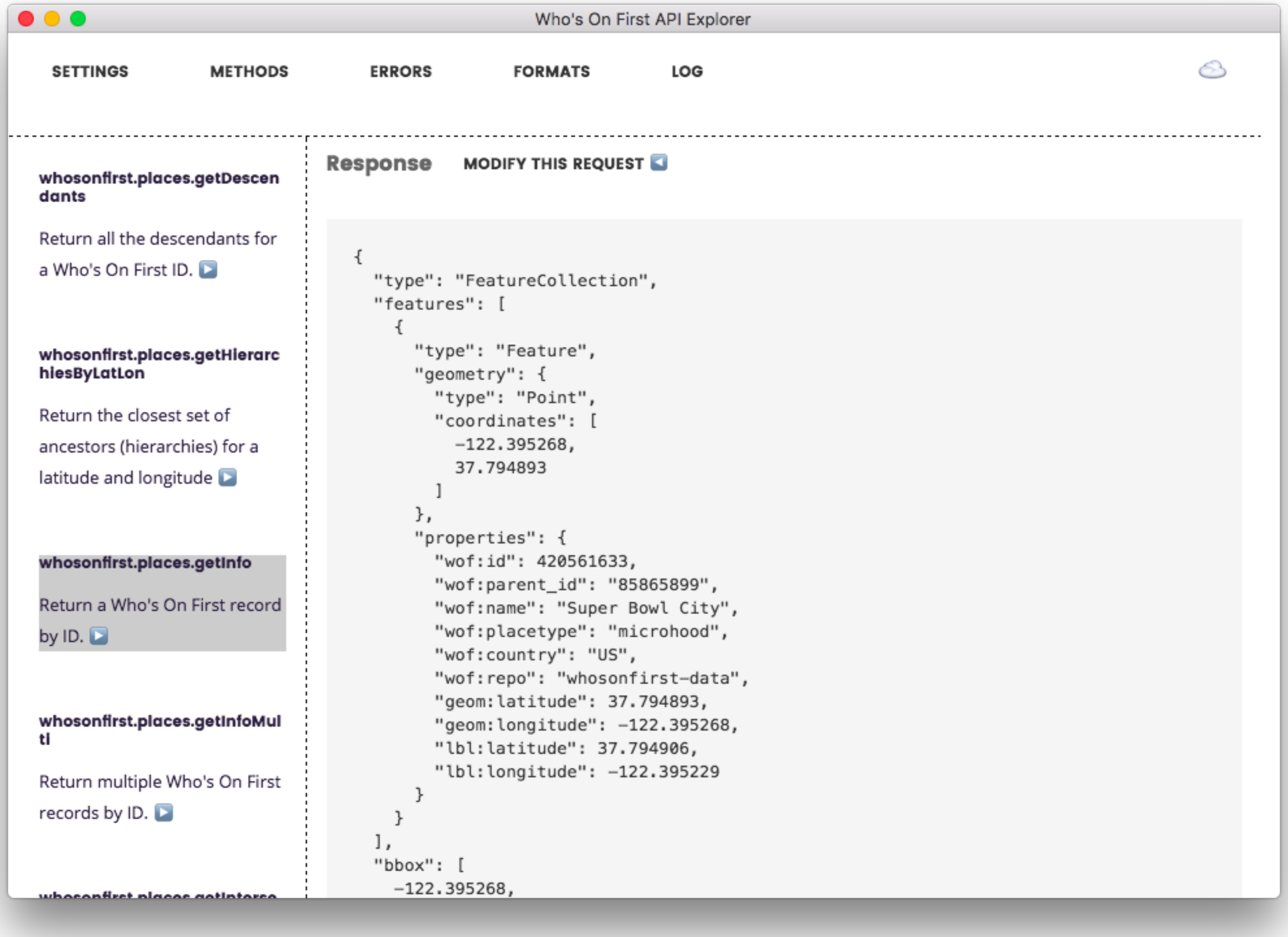
Return multiple Who's On First records by ID.

whosonfirst.places.getInfo

Request MODIFY THIS REQUEST

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?
method=whosonfirst.places.getInfo&api_key=mapzen-xxxxxx&id=420561633&format=geojson'
```

:partyparrot: (https://github.com/whosonfirst/electron-whosonfirst-api-explorer/blob/master/images/party-parrot.gif) *will keep you company while your API request is being processed.*



whosonfirst.places.getDescendants

Return all the descendants for a Who's On First ID.

whosonfirst.places.getHierarchiesByLatLon

Return the closest set of ancestors (hierarchies) for a latitude and longitude

whosonfirst.places.getInfo

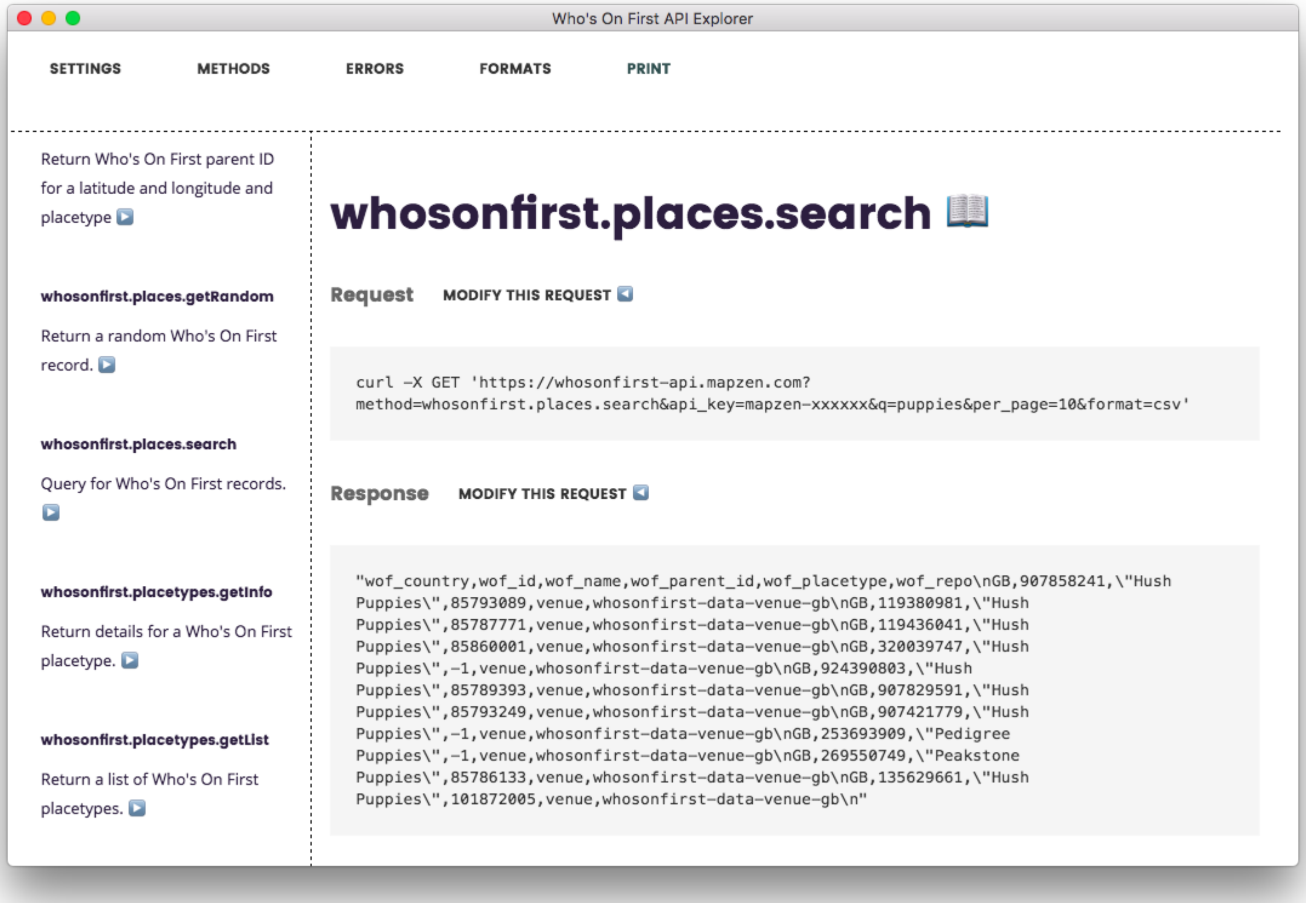
Return a Who's On First record by ID.

whosonfirst.places.getInfoMulti

Return multiple Who's On First records by ID.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [
          -122.395268,
          37.794893
        ]
      },
      "properties": {
        "wof:id": 420561633,
        "wof:parent_id": "85865899",
        "wof:name": "Super Bowl City",
        "wof:placetype": "microhood",
        "wof:country": "US",
        "wof:repo": "whosonfirst-data",
        "geom:latitude": 37.794893,
        "geom:longitude": -122.395268,
        "lbl:latitude": 37.794906,
        "lbl:longitude": -122.395229
      }
    }
  ],
  "bbox": [
    -122.395268,
```

Oh look, here's a new part of the Who's On First API: ***GeoJSON (https://mapzen.com/documentation/wof/formats/#geojson)*** *as an output format.*



Return Who's On First parent ID for a latitude and longitude and placetype

whosonfirst.places.getRandom

Return a random Who's On First record.

whosonfirst.places.search

Query for Who's On First records.

whosonfirst.placetypes.getInfo

Return details for a Who's On First placetype.

whosonfirst.placetypes.getList

Return a list of Who's On First placetypes.

whosonfirst.places.search 📖

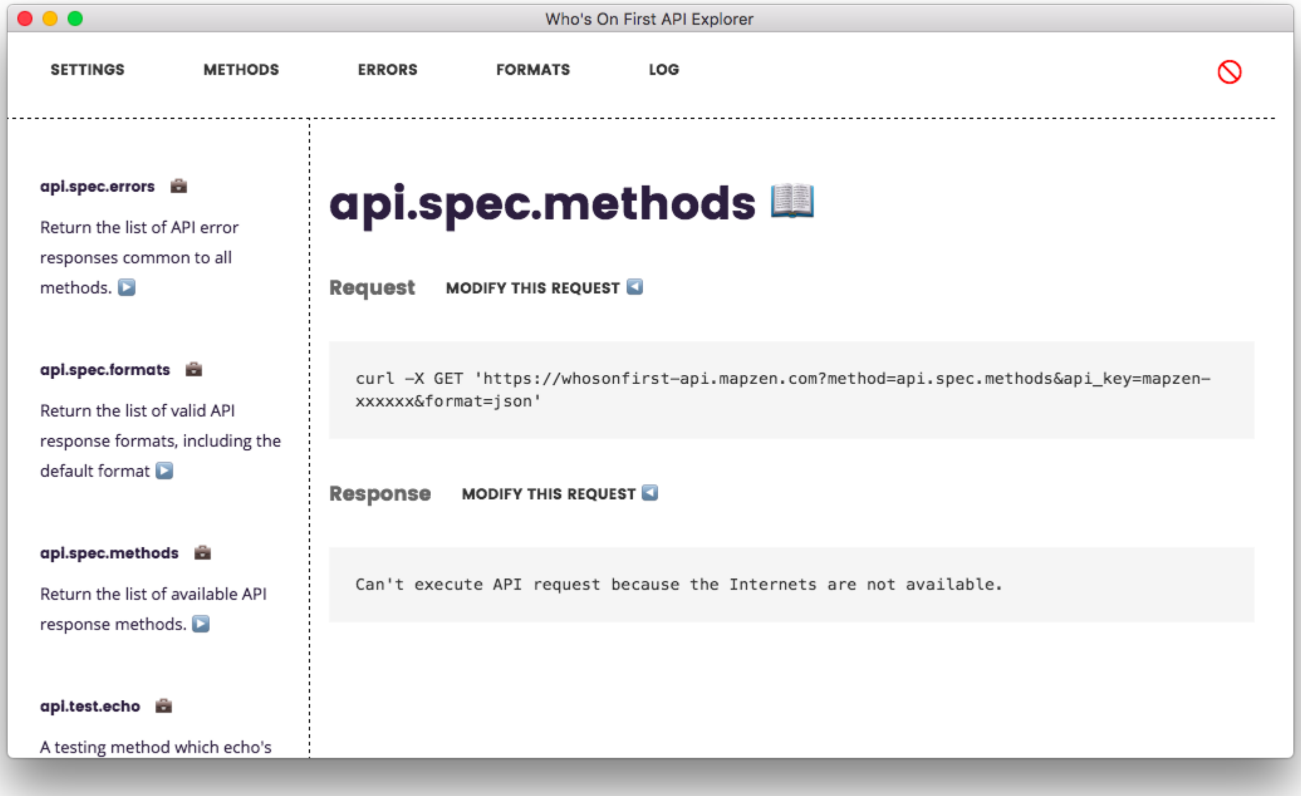
Request

```
curl -X GET 'https://whosonfirst-api.mapzen.com?
method=whosonfirst.places.search&api_key=mapzen-xxxxxx&q=puppies&per_page=10&format=csv'
```

Response

```
"wof_country,wof_id,wof_name,wof_parent_id,wof_placetype,wof_repo\nGB,907858241,\"Hush
Puppies\",85793089,venue,whosonfirst-data-venue-gb\nGB,119380981,\"Hush
Puppies\",85787771,venue,whosonfirst-data-venue-gb\nGB,119436041,\"Hush
Puppies\",85860001,venue,whosonfirst-data-venue-gb\nGB,320039747,\"Hush
Puppies\",-1,venue,whosonfirst-data-venue-gb\nGB,924390803,\"Hush
Puppies\",85789393,venue,whosonfirst-data-venue-gb\nGB,907829591,\"Hush
Puppies\",85793249,venue,whosonfirst-data-venue-gb\nGB,907421779,\"Hush
Puppies\",-1,venue,whosonfirst-data-venue-gb\nGB,253693909,\"Pedigree
Puppies\",-1,venue,whosonfirst-data-venue-gb\nGB,269550749,\"Peakstone
Puppies\",85786133,venue,whosonfirst-data-venue-gb\nGB,135629661,\"Hush
Puppies\",101872005,venue,whosonfirst-data-venue-gb\n"
```

The API Explorer can also render non-JSON output formats. There is also ***an open ticket to enable downloading of results (https://github.com/whosonfirst/electron-whosonfirst-api-explorer/issues/21)*** *from the API Explorer.*



api.spec.errors 💼

Return the list of API error responses common to all methods. ▶

api.spec.formats 💼

Return the list of valid API response formats, including the default format ▶

api.spec.methods 💼

Return the list of available API response methods. ▶

api.test.echo 💼

A testing method which echo's

api.spec.methods 📖

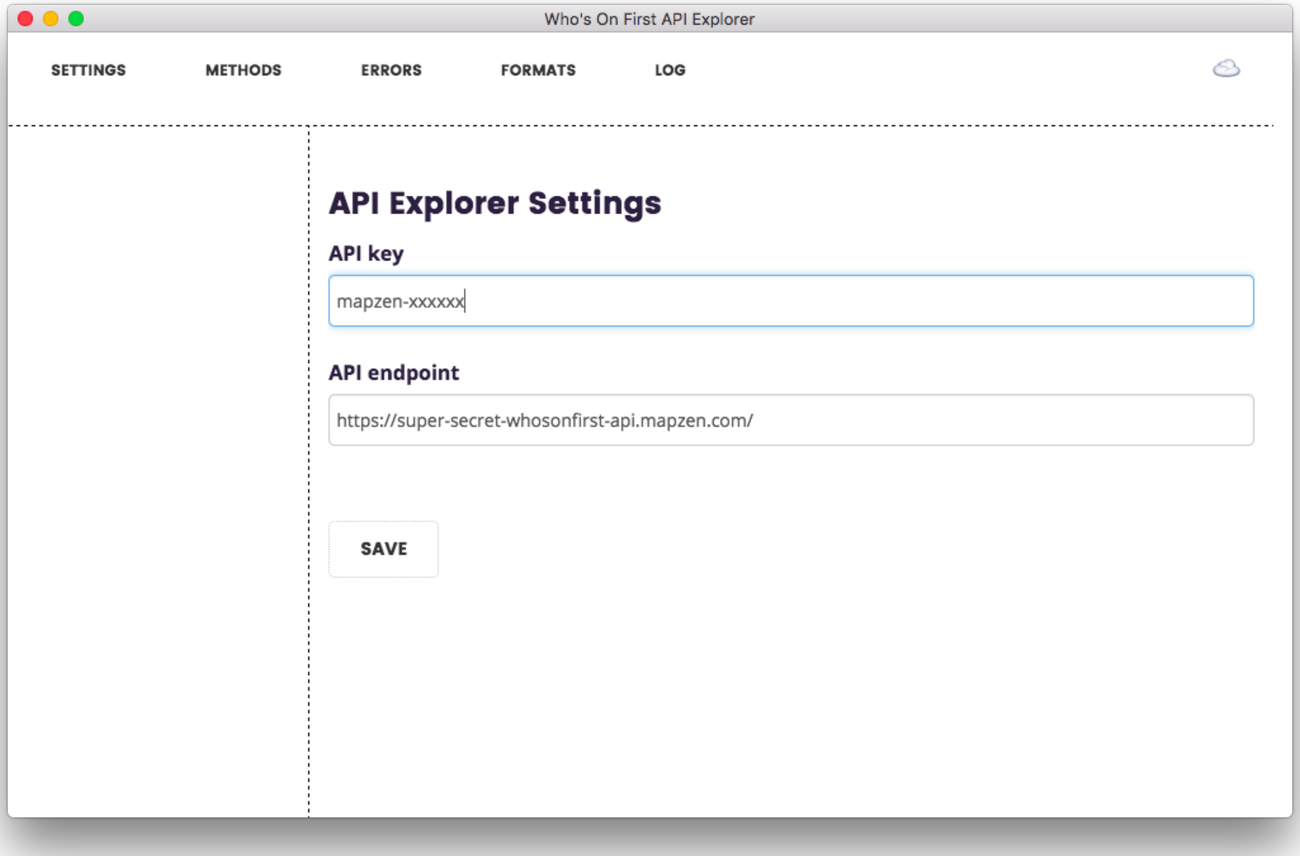
Request MODIFY THIS REQUEST ◀

```
curl -X GET 'https://whosonfirst-api.mapzen.com?method=api.spec.methods&api_key=mapzen-
xxxxxx&format=json'
```

Response MODIFY THIS REQUEST ◀

```
Can't execute API request because the Internets are not available.
```

Importantly the API Explorer will continue to work offline. See the way the ☁️ in the top-left corner became a 🚫 - that means the API Explorer can’t find the Internets. You won’t be able to send API requests out across the network but you can still read the documentation and construct API requests to review.



API Explorer Settings

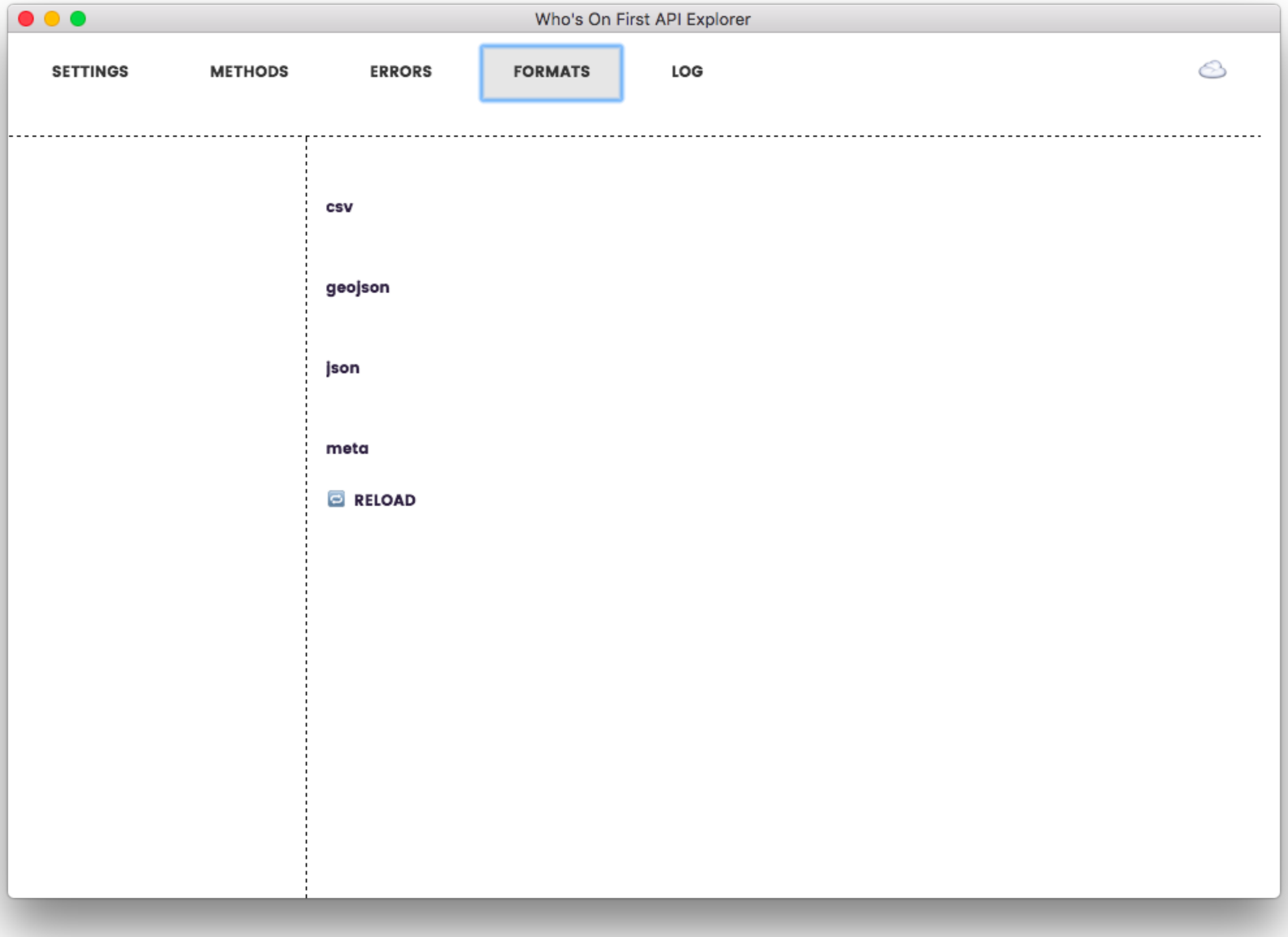
API key

mapzen-xxxxxx

API endpoint

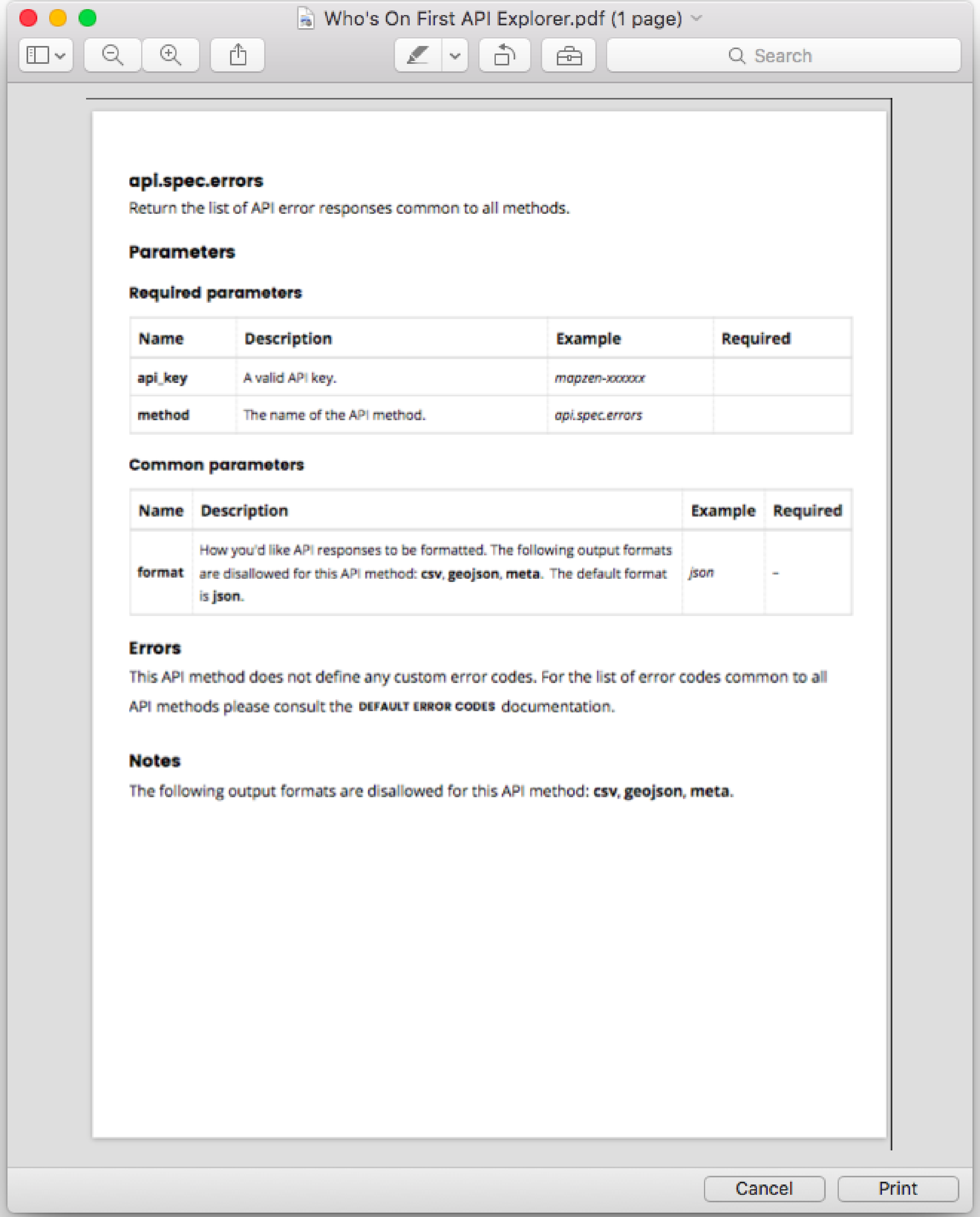
https://super-secret-whosonfirst-api.mapzen.com/

You can also change the API endpoint for making requests and by extension information about the API itself. via the `api.spec` *methods. This probably isn't that useful to anyone else right now but it's very helpful for us when testing new API methods on our dev servers. Eventually* ***we'd like to support "named" and "toggle-able" configurations (https://github.com/whosonfirst/electron-whosonfirst-api-explorer/issues/31)*** *in the API Explorer to make it easy to switch from one API endpoint to another.*


Who's On First API Explorer
SETTINGS
METHODS
ERRORS
FORMATS
LOG
csv
geojson
json
meta
RELOAD

Most of the work, in both the API and the Explorer, has been around API methods. There is still some more work to do for the other things…

There's an ***an experimental and currently disabled print feature (https://github.com/whosonfirst/electron-whosonfirst-api-explorer/issues/10)*** *to generate print-friendly version of both API documentation and results.*



api.spec.errors

Return the list of API error responses common to all methods.

Parameters

Required parameters

Name	Description	Example	Required
api_key	A valid API key.	*mapzen-xxxxxx*	
method	The name of the API method.	*api.spec.errors*	

Common parameters

Name	Description	Example	Required
format	How you'd like API responses to be formatted. The following output formats are disallowed for this API method: **csv**, **geojson**, **meta**. The default format is **json**.	*json*	-

Errors

This API method does not define any custom error codes. For the list of error codes common to all API methods please consult the **DEFAULT ERROR CODES** documentation.

Notes

The following output formats are disallowed for this API method: **csv**, **geojson**, **meta**.

Unfortunately, I can't figure out why print versions are truncated after one "page" so we'll turned off that functionality for now. Also apparently Emoji (specifically the 👍signifying a required parameter) are silently dropped in the print output because… computers?

Like the API itself, the API Explorer still tilts towards beta as of this writing. There are a number of **outstanding issues (https://github.com/whosonfirst/electron-whosonfirst-api-explorer/issues)**, most of them minor and nothing that should prevent you from using the API Explorer from… exploring the **Who's On First API (https://mapzen.com/documentation/wof)**.

The source code is available from **https://github.com/whosonfirst/electron-whosonfirst-api-explorer (https://github.com/whosonfirst/electron-whosonfirst-api-explorer)** and we would welcome your feedback and suggestions.

Image credits:

- **Drawing, To Explore by that Cheerless Illumination for Leaks, ca. 1904; Frank Walter Taylor ; USA; point of brush and black, gray wash, white goache, black crayon on bristol board; Sheet: 44.4 x 30.6 cm (17 ½ x 12 1/16 in.); Gift of F. Elizabeth De Voy; 1957-58-15 (https://collection.cooperhewitt.org/objects/18419753/)**

· 28 April 2017 ·

Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

May Day Mapping

flex (/tag/flex) vector-tiles (/tag/vector-tiles) routing (/tag/routing) search (/tag/search)

Mappers of the world, unite! You have nothing to lose but your too-complicated pricing tiers. All apps are equal! All users are equal!

Last week we announced **Mapzen Flex (https://mapzen.com/pricing/)**, the new way developers can access Mapzen’s vector tiles, geocoding, and turn-by-turn navigation services beyond our free rate limits. Reaction from the open mapping community was swift:

> holy crap Mapzen started actually selling things to people **https://t.co/h9JhfyzPq0 (https://t.co/h9JhfyzPq0)**
>
> — Anonymaps (@Anonymaps) **April 26, 2017 (https://twitter.com/Anonymaps/status/857203318774542337)**

To some extent, yes, Mapzen Flex is about Mapzen selling things to people. But if you look closer at the product you'll notice a few important elements at work. This is not the usual way mapping providers sell their services.

No tiers

First, we wanted to minimize confusion by charging users only for what they use. Generally you'd buy a plan for $50, $100, or some other even number, and stay within usage bounds. If you need more, you'd pay for the higher plan. This invites a business concept known as "breakage," which basically means "charging a user for things they don't use." We wanted to avoid this to be more accessible for all users, whether their budgets are big or small.

Per-request pricing

The ad industry has a concept called "CPM pricing," which can be confusing. It means "cost per thousand," and is often used to measure ad impressions. The M stands for *mille*, which is Latin for "thousand." We decided to price all our services like this because it seemed more predictable, scalable, and fair. It only costs us so much for a request, so why should the cost vary for you? (Don't worry, we only charge you for the part of the thousand that you use.)

Monthly budget caps

Long ago when I worked on the ad product that predated AdWords at Google, I learned not only about CPM pricing, but also the importance of budget capping for users who are spending money with you. So Mapzen Flex offers this too: you can set a budget you don't want to exceed in a given month. We'll let you know if you're close and you can decide what to do, but the control is in your hands.

Same price for your free and paid and mobile users

We didn't want to charge different prices for different user or mobile app types. Since a request to Mapzen has a fixed cost, it doesn't seem reasonable to vary that cost to our users based on what they're doing. We want to serve as many requests as possible and don't see why you should be restricted here.

Sound good? **Here's our pricing (https://mapzen.com/pricing/)**. We offer generous free rate limits and as of June 1 you'll need a credit card saved to your account to go above them. But the pricing will be predictable, accessible, and fair. **Let us know if you have questions here (mailto:pricing@mapzen.com)**!

Image via **Cooper Hewitt (https://collection.cooperhewitt.org/objects/18638827/)**, *Poster, The First Day of May, ca. 1920; Designed by Nikolai Kogout (1891–1959); Russia; color lithograph on newsprint; 71 x 53.6 cm (27 15/16 x 21 1/8 in.); Museum purchase through gift of Mrs. John Innes Kane; 1992-123-3*

· 01 May 2017 ·

Randy Meech

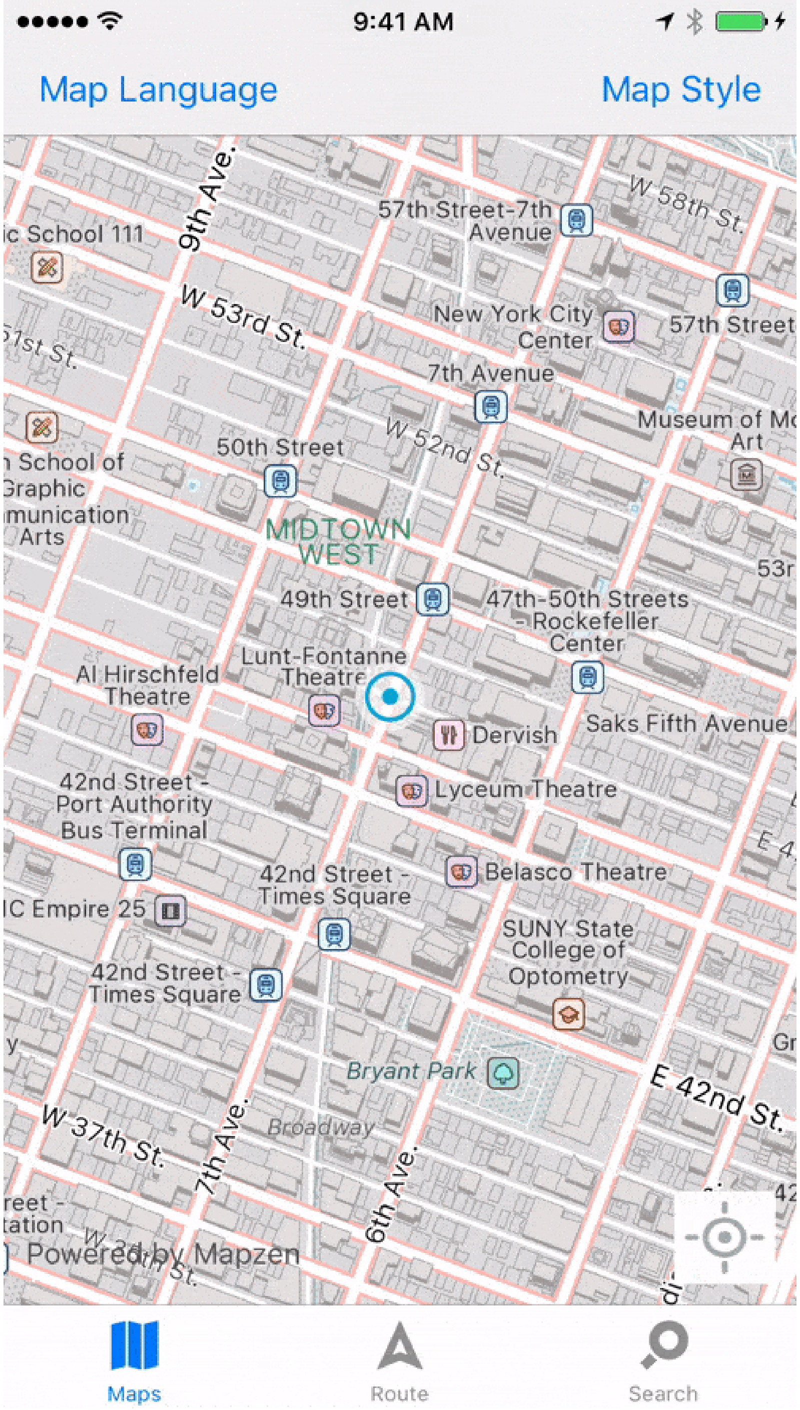
Billerica Memorial High School graduate. BA, MTS. Mapzen CEO 2013-present.

© 2017 Mapzen

Announcing the Mapzen iOS SDK

mobile (/tag/mobile)

Today we are launching **version 1.0.0 of the Mapzen iOS SDK (https://cocoapods.org/pods/Mapzen-ios-sdk)**. This release brings search, routing, and map rendering to iOS. You can now easily add markers to a map, manipulate the position, rotation, and zoom of the map, switch map styles and languages along with many more features. For a full list of what you can do, take a look at the **documentation (https://mapzen.com/documentation/ios/)** or the **GitHub page (https://github.com/mapzen/ios)**.


9:41 AM
Map Language
Map Style
9th Ave.
57th Street-7th Avenue
W 58th St.
W 53rd St.
New York City Center
57th Street
7th Avenue
Museum of Mo Art
50th Street
W 52nd St.
MIDTOWN WEST
49th Street
47th-50th Streets - Rockefeller Center
Lunt-Fontanne Theatre
Al Hirschfeld Theatre
Dervish
Saks Fifth Avenue
Lyceum Theatre
42nd Street - Port Authority Bus Terminal
Belasco Theatre
42nd Street - Times Square
SUNY State College of Optometry
Bryant Park
E 42nd St.
Broadway
W 37th St.
7th Ave.
6th Ave.
Powered by Mapzen
Maps
Route
Search

The iOS SDK is written using Swift version 3.0 and is fully compatible with Objective-C. It relies on these stand alone projects from Mapzen:

- **Tangram (https://github.com/tangrams/tangram-es)**
- **Pelias (https://github.com/pelias/pelias-ios-sdk)**
- **On The Road (https://github.com/mapzen/on-the-road_ios)**

The rest of this week we'll be posting everyday about a different part of the Mapzen iOS SDK and how it can be helpful to your location based apps. We hope you'll follow along!

- **Tuesday (5/2)** Announcing the Mapzen iOS SDK
- **Wednesday (5/3)** Rendering maps on iOS with Tangram
- **Thursday (5/4)** Find your place with Mapzen Search for iOS
- **Friday (5/5)** Routing on iOS with Mapzen Turn-by-Turn and On the Road

Give the SDK a try and let us know what you like and don't like. We'd also love to hear about features you want to see in future releases. Email us at **ios-support@mapzen.com (mailto:ios-support@mapzen.com)** to get in touch.

· 02 May 2017 ·

Sarah Lensing

mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

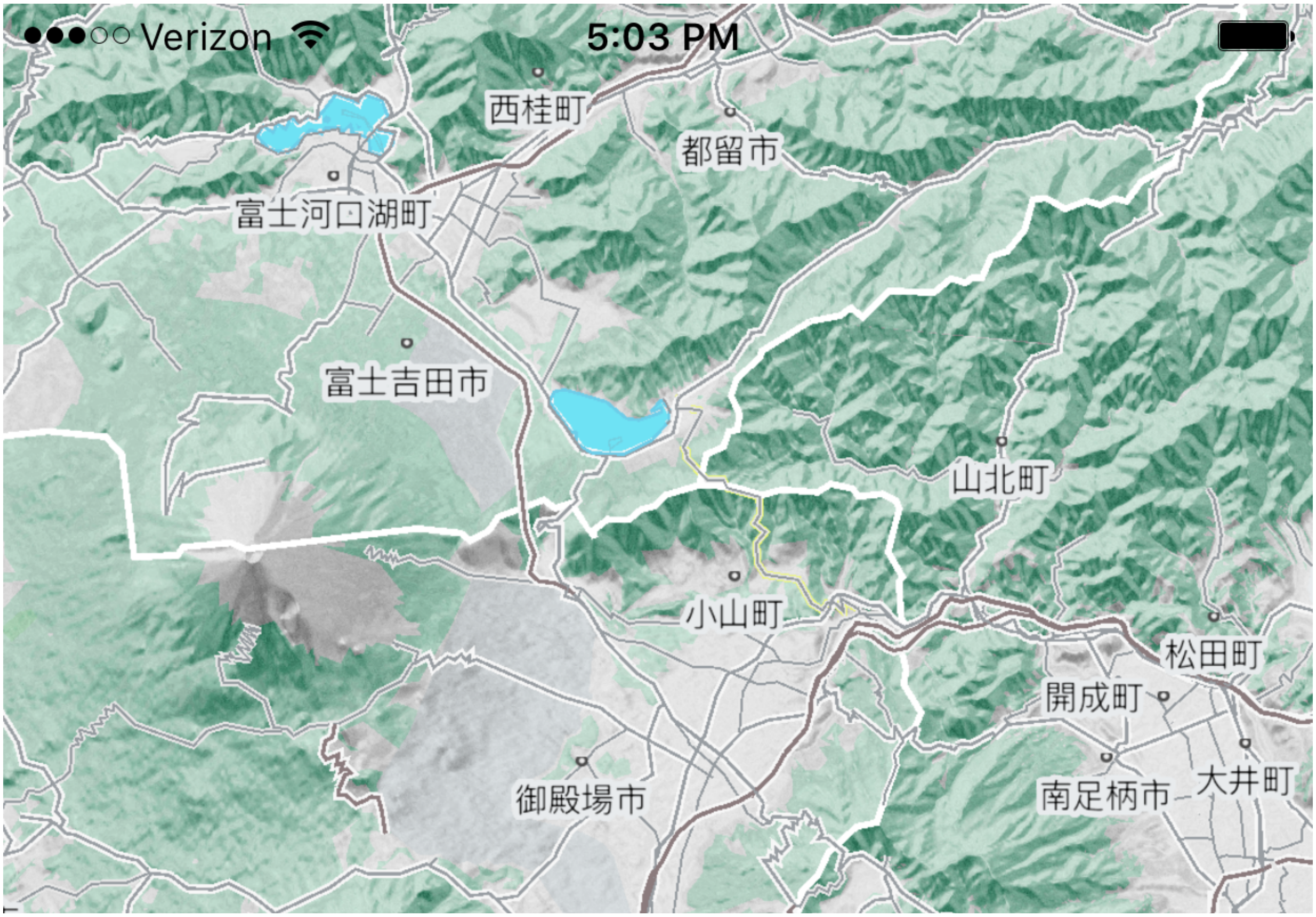
© 2017 Mapzen

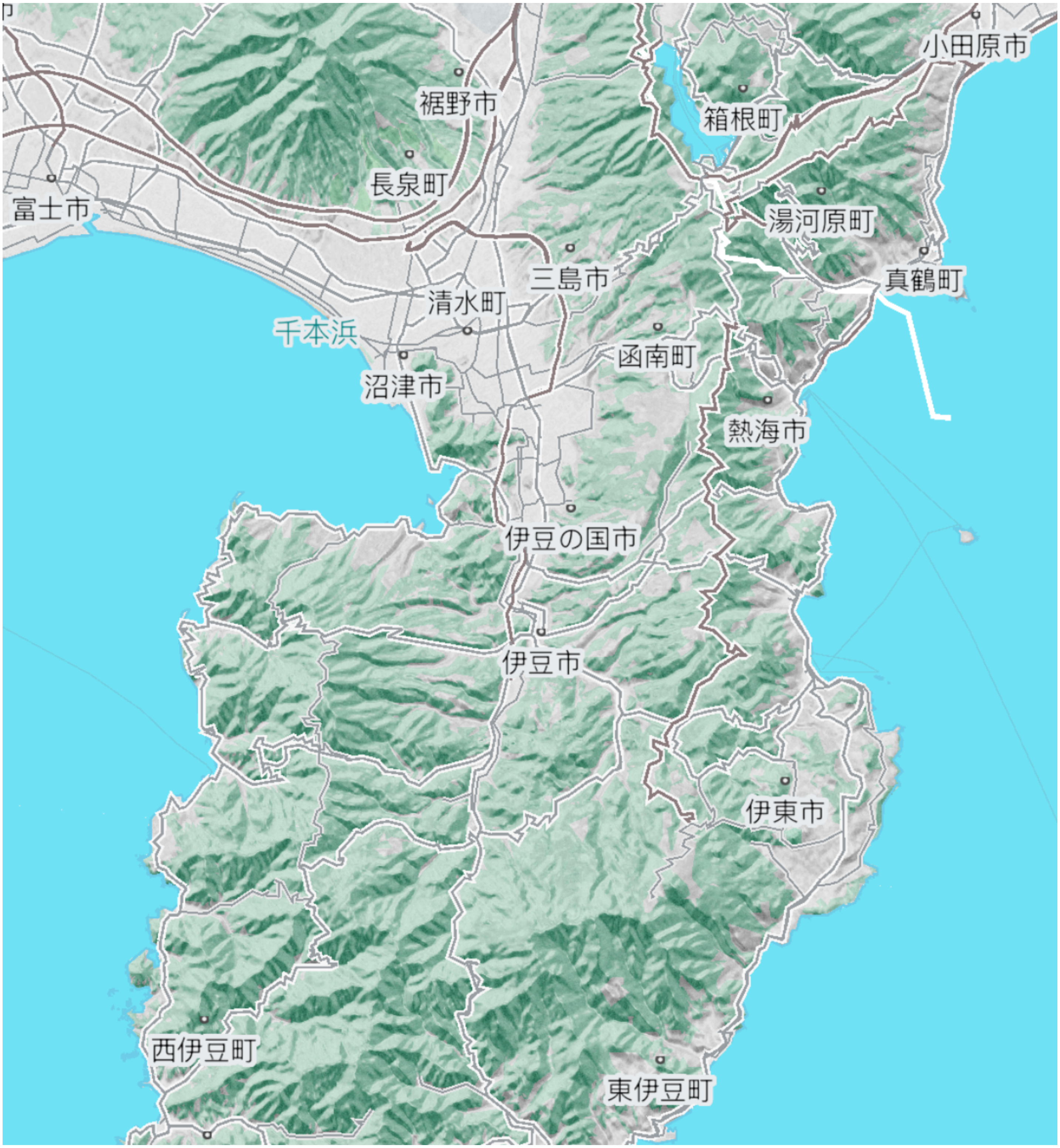
Rendering maps on iOS with Tangram

tangram (/tag/tangram) mobile (/tag/mobile)

One component of the Mapzen iOS SDK is Tangram, a rendering engine designed to efficiently paint highly customizable maps from a handful of agnostic data sources. It has the responsibility of drawing markers for routes, icons, and the visually pleasing and effective basemap styles that our expert cartography team designs.

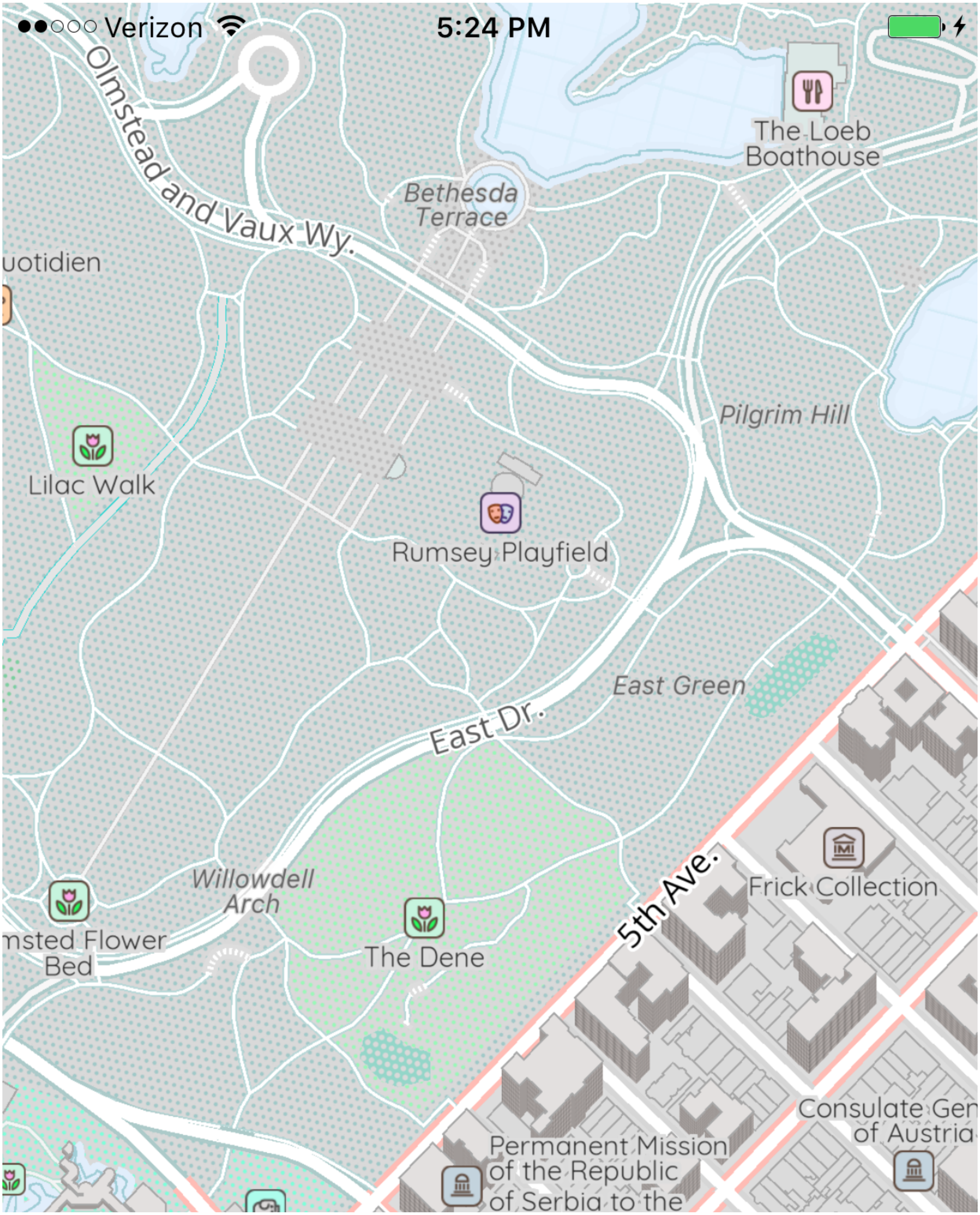
Tangram as a cartographic technology solves complex map label rendering and placement, and shades elevation data in a rather unique way to make stunning-looking terrain. It will bend your map labels so they look their best on curved paths, and draws text with a crisp quality, handling **many different script features (http://r12a.github.io/scripts/featurelist/)**. If you are interested in learning more about those features, or are just curious about Tangram, please take a look at its **documentation (https://mapzen.com/documentation/tangram/)**.



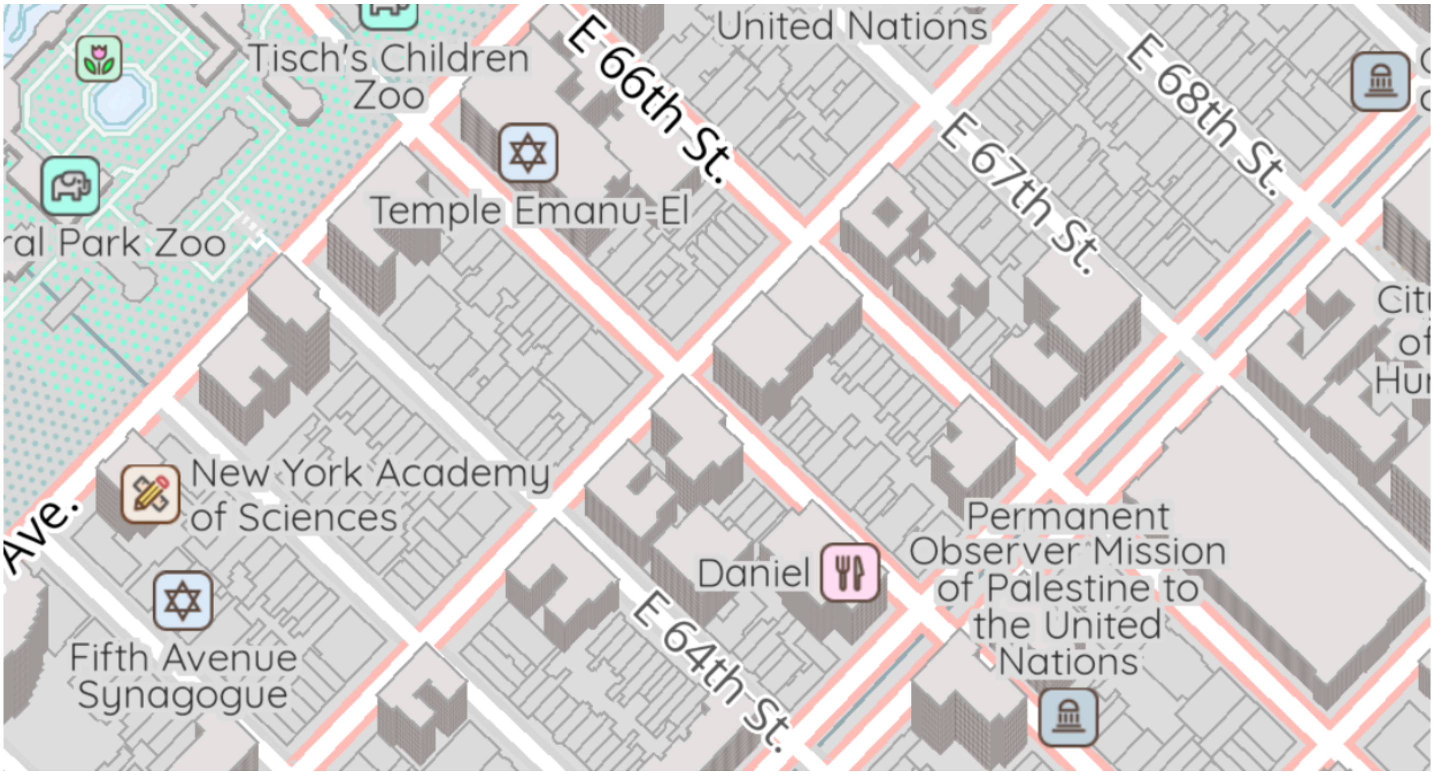

小田原市
裾野市
箱根町
長泉町
富士市
湯河原町
三島市
真鶴町
清水町
千本浜
函南町
沼津市
熱海市
伊豆の国市
伊豆市
伊東市
西伊豆町
東伊豆町

This **platform-agnostic map rendering engine (https://github.com/tangrams/tangram-es)** fits the needs of our iOS SDK for several reasons. We made it relatively straightforward to integrate Tangram so it can render maps in any iOS application or framework. The engine is entirely open-source under the permissive MIT license and, even though written in C++ in its core, it has support for both Objective-C and Swift.

The styling system and its efficacious flexibility lets you use the home-crafted and delightful **Mapzen cartography (https://mapzen.com/products/maps/)** out of the box from the iOS SDK, where the different data sources that Mapzen provides will look great on your devices.



●●○○○ Verizon
5:24 PM
Olmstead and Vaux Wy.
The Loeb Boathouse
Bethesda Terrace
uotidien
Pilgrim Hill
Lilac Walk
Rumsey Playfield
East Green
East Dr.
5th Ave.
Frick Collection
Willowdell Arch
msted Flower Bed
The Dene
Consulate Gen of Austria
Permanent Mission of the Republic of Serbia to the


United Nations
Tisch's Children Zoo
E 66th St.
E 68th St.
E 67th St.
Temple Emanu-El
ral Park Zoo
New York Academy of Sciences
Ave.
Daniel
Permanent Observer Mission of Palestine to the United Nations
E 64th St.
Fifth Avenue Synagogue

The iOS SDK is a merging point of several mapping technologies and aims to make it easy to use Tangram for rendering among several other services like routing, geocoding and search. It makes your map fast to set up, and ready to work and interact with, giving you more time to think about the other nice things in your iOS application instead.

This Swiss Army knife becomes really handy, and even if Tangram is that shiny bottle opener that all of your friends are asking for, you probably want to know more about all those other technologies that comes along when using it.

Getting a map view into your app with the Mapzen iOS SDK is straightforward and should be familiar if you've used other mapping SDKs. We've documented it all here: **https://mapzen.com/documentation/ios/getting-started/ (https://mapzen.com/documentation/ios/getting-started/)**

If full control and flexibility over the map rendering is what you need, our Tangram iOS Framework is also available for use directly from CocoaPods as **a binary framework (https://cocoapods.org/pods/Tangram-ES)**, and a walkthrough is available for you to get started. **https://mapzen.com/documentation/tangram/iOS-walkthrough/ (https://mapzen.com/documentation/tangram/iOS-walkthrough/)**.

Stay tuned all this week as we post about a different part of the Mapzen iOS SDK every day and how it can be helpful to your location based apps. If you missed the first post in this series you can go back and check it out now.

- **Tuesday (5/2)** Announcing the Mapzen iOS SDK
- **Wednesday (5/3)** Rendering maps on iOS with Tangram
- **Thursday (5/4)** Find your place with Mapzen Search for iOS
- **Friday (5/5)** Routing on iOS with Mapzen Turn-by-Turn and On the Road

Let us know (mailto:ios-support@mapzen.com) what apps you make with Tangram on iOS!

· 03 May 2017 ·

Karim Naaji

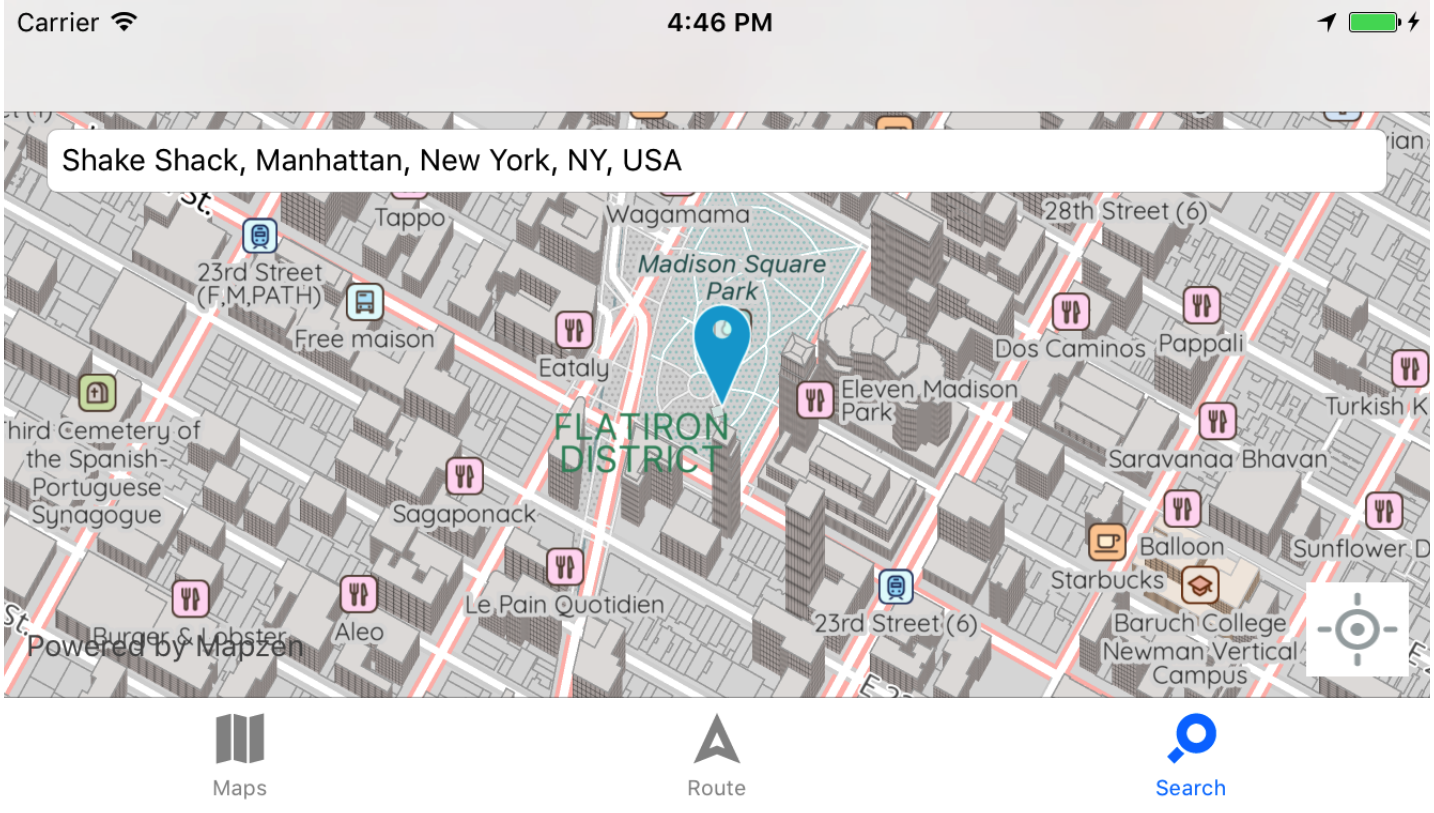
© 2017 Mapzen

Find your place with Mapzen Search for iOS

mobile (/tag/mobile)

Back in October we launched Mapzen Search on Android. We are excited to tell you that today Mapzen Search is now available on iOS.

Mapzen Search is powered by entirely open data from a few high quality **sources (https://mapzen.com/documentation/search/data-sources/)**. Features of this service include forward geocoding, reverse geocoding, and autocomplete. With the results from any of these queries, you can fetch details on the location using the place endpoint.



Shake Shack, Manhattan, New York, NY, USA

Carrier
4:46 PM
Tappo
Wagamama
Madison Square Park
28th Street (6)
23rd Street (F,M,PATH)
Free maison
Eataly
Dos Caminos
Pappali
Eleven Madison Park
FLATIRON DISTRICT
Third Cemetery of the Spanish-Portuguese Synagogue
Turkish K
Saravanaa Bhavan
Sagaponack
Balloon
Sunflower D
Starbucks
Le Pain Quotidien
23rd Street (6)
Baruch College Newman Vertical Campus
Burger & Lobster
Aleo
Powered by Mapzen
Maps
Route
Search

Mapzen iOS SDK demo app

Like Mapzen Search uses the **Pelias (https://github.com/pelias/pelias)** project, Mapzen Search for iOS uses the **Pelias iOS (github.com/pelias/pelias-ios-sdk)** project. Take a look at the **documentation (https://mapzen.com/documentation/ios/)** to help get you started or checkout the **sample app (https://github.com/mapzen/ios/tree/master/SampleApp)** for a working example.

Stay tuned all this week as we post about a different part of the Mapzen iOS SDK every day and how it can be helpful to your location based apps. If you missed the earlier posts in this series you can go back and check them out!

- **Tuesday (5/2)** Announcing the Mapzen iOS SDK
- **Wednesday (5/3)** Rendering maps on iOS with Tangram
- **Thursday (5/4)** Find your place with Mapzen Search for iOS
- **Friday (5/5)** Routing on iOS with Mapzen Turn-by-Turn and On the Road

As always, we would love to hear from you. Tell us about what you are building or submit issues **here (https://github.com/mapzen/ios/issues/new)**.

· 04 May 2017 ·

Sarah Lensing

mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

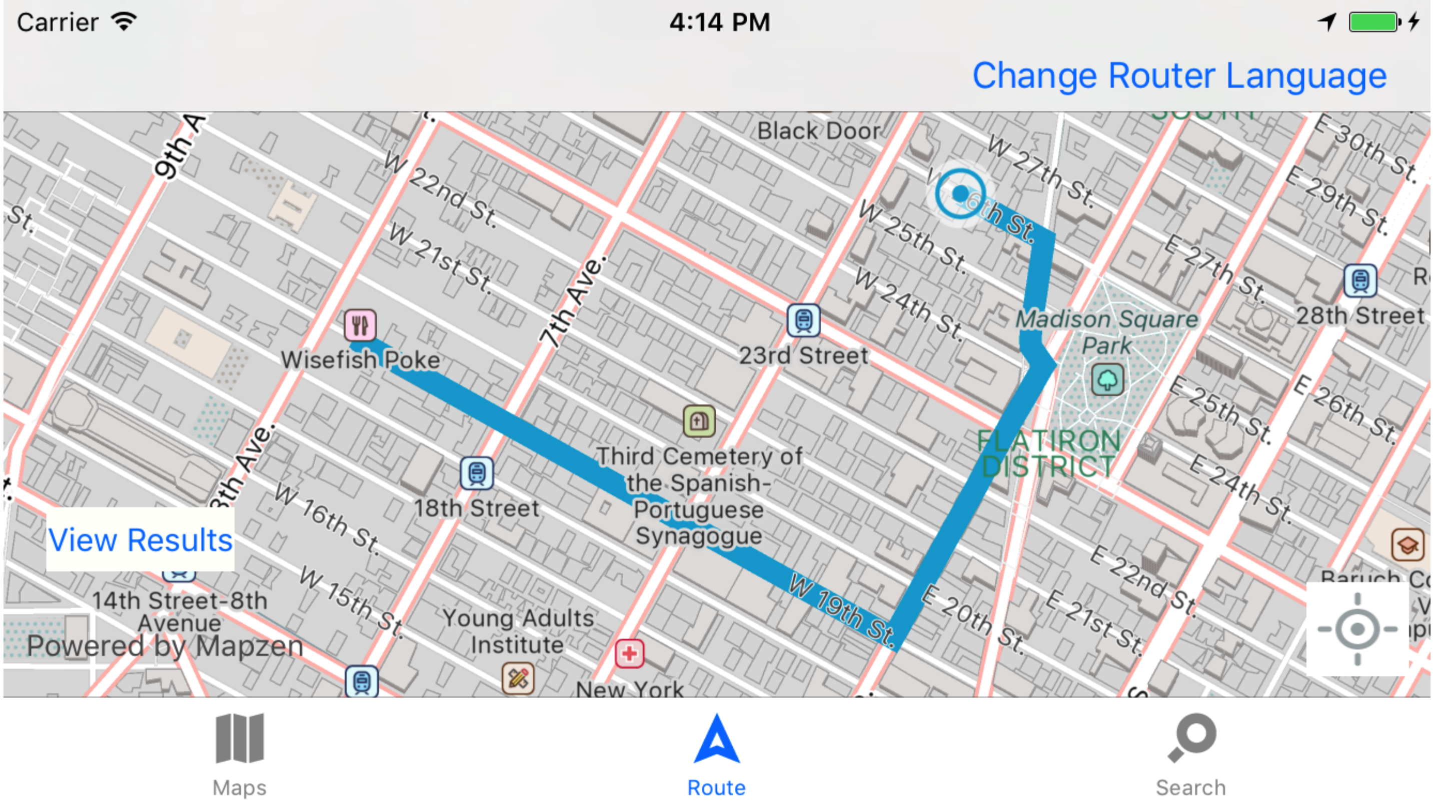
© 2017 Mapzen

Routing on iOS with Mapzen Turn-by-Turn and On the Road

mobile (/tag/mobile)

We are excited to tell you about the launch of Mapzen Turn-by-Turn on iOS as part of the **Mapzen iOS SDK (https://github.com/mapzen/ios)**.

One of the three main pieces of the Mapzen iOS SDK, Mapzen Turn-by-Turn provides driving directions and customizable route lines using our hosted routing service, powered by **Valhalla (https://github.com/valhalla/valhalla)**.


Carrier
4:14 PM
Change Router Language
Black Door
W 27th St.
W 22nd St.
W 21st St.
7th Ave.
W 25th St.
W 24th St.
Madison Square Park
E 27th St.
28th Street
E 30th St.
E 29th St.
Wisefish Poke
23rd Street
FLATIRON DISTRICT
E 25th St.
E 26th St.
8th Ave.
Third Cemetery of the Spanish-Portuguese Synagogue
18th Street
W 16th St.
View Results
W 19th St.
E 24th St.
E 22nd St.
14th Street-8th Avenue
W 15th St.
Young Adults Institute
E 20th St.
E 21st St.
Powered by Mapzen
New York
Maps
Route
Search

Mapzen Turn-by-Turn for iOS uses the **OnTheRoad for iOS (https://github.com/mapzen/on-the-road_ios)** project, which is our open source library for connecting to a **Valhalla (https://github.com/valhalla/valhalla)** server. Take a look at the **documentation (https://mapzen.com/documentation/ios/)** to help get you started or checkout the **sample app (https://github.com/mapzen/ios/tree/master/SampleApp)** for a working routing example.

All this week as we have been posting about different parts of the Mapzen iOS SDK and how it can be helpful to your location based apps. If you missed the earlier posts in this series you can go back and check them out now!

- **Tuesday (5/2)** Announcing the Mapzen iOS SDK
- **Wednesday (5/3)** Rendering maps on iOS with Tangram
- **Thursday (5/4)** Find your place with Mapzen Search for iOS
- **Friday (5/5)** Routing on iOS with Mapzen Turn-by-Turn and On the Road

As always, we would love to hear from you. **Tell us about what you are building (mailto:hello@mapzen.com)** or submit issues **here (https://github.com/mapzen/ios/issues/new)**.

· 05 May 2017 ·

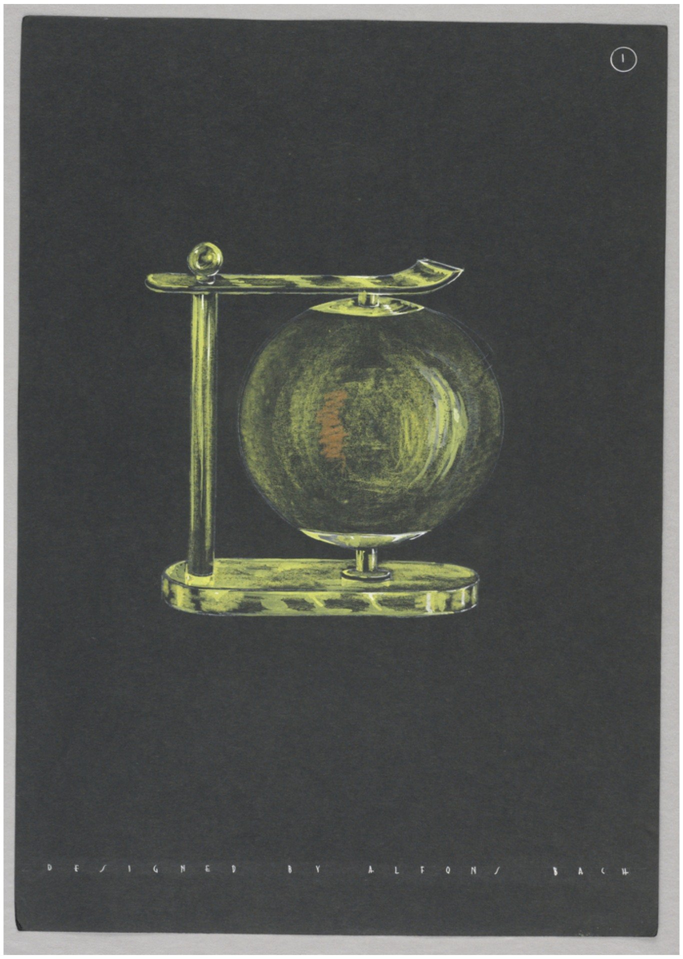
Matt Smollinger

Matt would 3D print an SDK if it were possible.

© 2017 Mapzen

Coarse Reverse Geocoding

search (/tag/search)



(https://collection.cooperhewitt.org/objects/18642873/)

Good news, everyone! The Mapzen Pelias team just released the first of several updates to our **/reverse endpoint (https://mapzen.com/documentation/search/reverse/)** that moves functionality away from what could be considered "things nearby" to true reverse geocoding. First is an update to "coarse reverse geocoding", or, searching for what administrative area(s) (city, state, country) contain a point. There's no trick to invoking **coarse reverse geocoding (https://mapzen.com/documentation/search/reverse/#filter-by-layers-data-type)** other than specifying a `layers` parameter with either `coarse` or any of the administrative area types, such as `locality`, `region`, or `country` in your `/v1/reverse` request.

A Quick Primer

The general idea of reverse geocoding is to determine the most granular geographical feature at a latitude/longitude pair for which there is data, be it an address, street, city, state, or country. The ideal data set for the target area (in our case, **the planet (https://whosonfirst.mapzen.com/spelunker/id/0)**) would contain a point for every single address, line strings with number ranges for every street, and polygons for every conceivable administrative area from microhoods to continents and oceans.

Some `/reverse` expectations:

- Appropriate thresholds are used in the case of addresses and streets to make sure things too far away are not presented as a match. A reverse geocoder populated with address data should return an address for a point if one is reasonably close (within, say, 25 meters).
- If no such address is found, it's expected that the reverse geocoder should fall back to a reasonably close interpolated address point on a line string representing a street.
- If no interpolated point is found, a reverse geocoder should fall back to a street where the input point is reasonably close to the line string.
- If no street exists, the most granular polygon representing administrative areas should be returned.

If sufficient data exists, a point on Earth should always return a result, assuming that the input is a valid latitude/longitude pair, representing a geographical feature anywhere from an address in an alley in **Tuscon, AZ (https://whosonfirst.mapzen.com/spelunker/id/85917235)** all the way to somewhere in the **Arctic Ocean (https://whosonfirst.mapzen.com/spelunker/id/404528705)**.

In the general case, it's expected that a reverse geocoder will return a single result but certain concessions can be made in the case of areas that are disputed or change hands on occasion, such as **Doumera Island (https://whosonfirst.mapzen.com/spelunker/id/85632665/)** and **Pheasant Island (https://en.wikipedia.org/wiki/Pheasant_Island)**, respectively.

Pelias supports reverse geocoding with 2 required parameters containing the latitude and longitude pair and an optional 3rd parameter specifying 1 or more layers. An example request that finds the city for point 33.141112/-107.249610 is:

```
http://search.mapzen.com/v1/reverse?layers=locality&point.lat=33.141112&point.lon=-107.249610
```

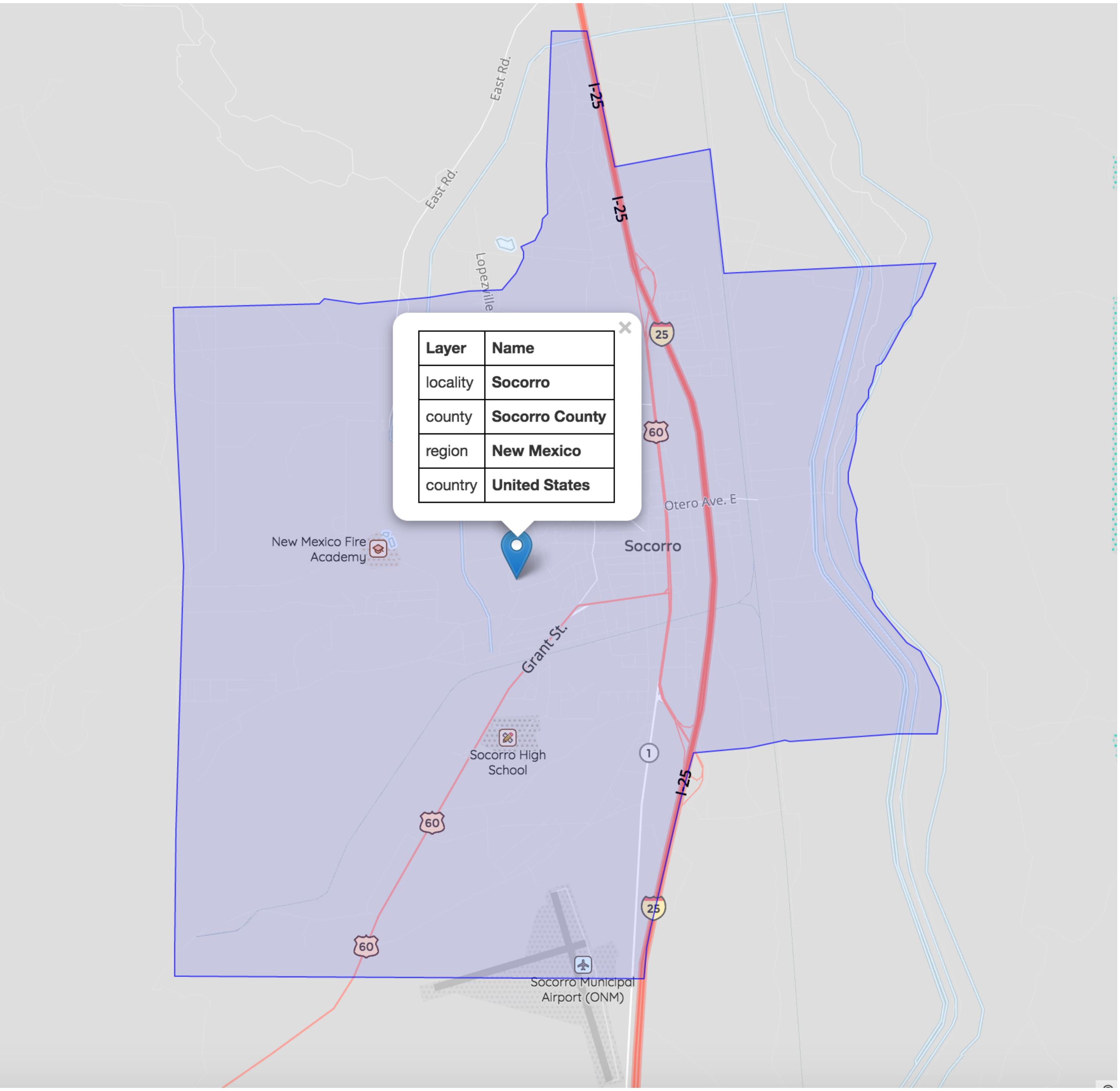
What We’ve Changed

We now consider if the point is in the geometry of the polygon, not just its distance to the centroid.

We’re rolling out the improvements in a few phases, starting with the case where the user just wants to know what administrative area a point is in. These would be requests specifying `layers=coarse` or any other value representing an administrative area, such as `neighborhood` or `country`. Specifying `venue`, `address`, or `street` or no parameter at all will disable the new logic and utilize the existing “what’s nearby” approach to reverse geocoding, which could be venues, addresses, or administrative areas.

This update searches administrative area polygons for an input point, rather than only returning an administrative area if its geographical centroid happened to be close to that point. Effectively, this change means that we’re now doing a simple point-in-polygon lookup for coarse `/reverse` requests.

For example, the known hierarchy of a point is returned:


I-25
East Rd.
East Rd.
I-25
Lopezville
25
60
Otero Ave. E
New Mexico Fire Academy
Socorro
Grant St.
Socorro High School
1
I-25
60
25
60
Socorro Municipal Airport (ONM)

Layer	Name
locality	**Socorro**
county	**Socorro County**
region	**New Mexico**
country	**United States**

The Near Future

The work described in this post can be viewed as fallback logic: find the most granular polygon that contains the point, be it a neighbourhood, city, or even country. The next step in updating how the `/reverse` endpoint works is to ensure that addresses returned are relevant to the input point. That is, only return an address result if there is one within a certain distance threshold of the input point. The new approach to coarse reverse geocoding fits nicely in with this plan since if an address (either explicit or interpolated) is not found, Pelias is already set up to fall back to the most granular polygon that contains the point.

The Far Future

We're in it for the long haul here at Mapzen, so even before Mars is colonized, Pelias should be able to reverse geocode on the red planet when there is a Martian gazetteer! (Note that `planet` is already a placetype in **Who's On First (https://whosonfirst.mapzen.com/spelunker/placetypes/planet)**…) In fact, the only stipulation we make for whatever planets you want to load into a custom Pelias instance is that they be as spherical as possible. (Sorry, Phobos.) We're not outright discriminating against non-sphere-ish astronomical bodies, we just can't map their coordinate systems right now. We'll cross that space bridge when we come to it.

Image credits:

- **Drawing, 1. Green globe table lamp; Designed by Alfons Bach (German, 1904–1999); USA; chalk, watercolor, wash, black paper; Gift of Alfons Bach; 1993-108-53 (https://collection.cooperhewitt.org/objects/18642873/)**

Note: This post was updated on May 31, 2017 to update the API request links.

· 09 May 2017 ·

Stephen Hess

Stephen works on geocoding exclusively as a means to fund his passion for designing and building wooden frames for old maps.

© 2017 Mapzen

Simple is hard

boundaryissues (/tag/boundaryissues) whosonfirst (/tag/whosonfirst) data (/tag/data)

Last Wednesday I gave a talk at **csv,conf (https://csvconf.com/)** in Portland, OR. The conference gathered a variety of people who care about open data, technical and non-technical alike. I adapted the talk from my last blog post **The world is weird and wonderful! (https://mapzen.com/blog/weird-and-wonderful/)** I've been told the conference organizers will post videos talks, so **stay tuned (https://twitter.com/csvconference)**.

Like the blog post, my talk was split into two parts. I started by introducing **Who's On First (https://mapzen.com/tag/whosonfirst/)** and **Boundary Issues (https://mapzen.com/tag/boundaryissues/)** to an audience not necessarily familiar with open geo, and then showed a hands-on demo of the **Boundary Issues CSV import feature (https://mapzen.com/blog/weird-and-wonderful/)**.

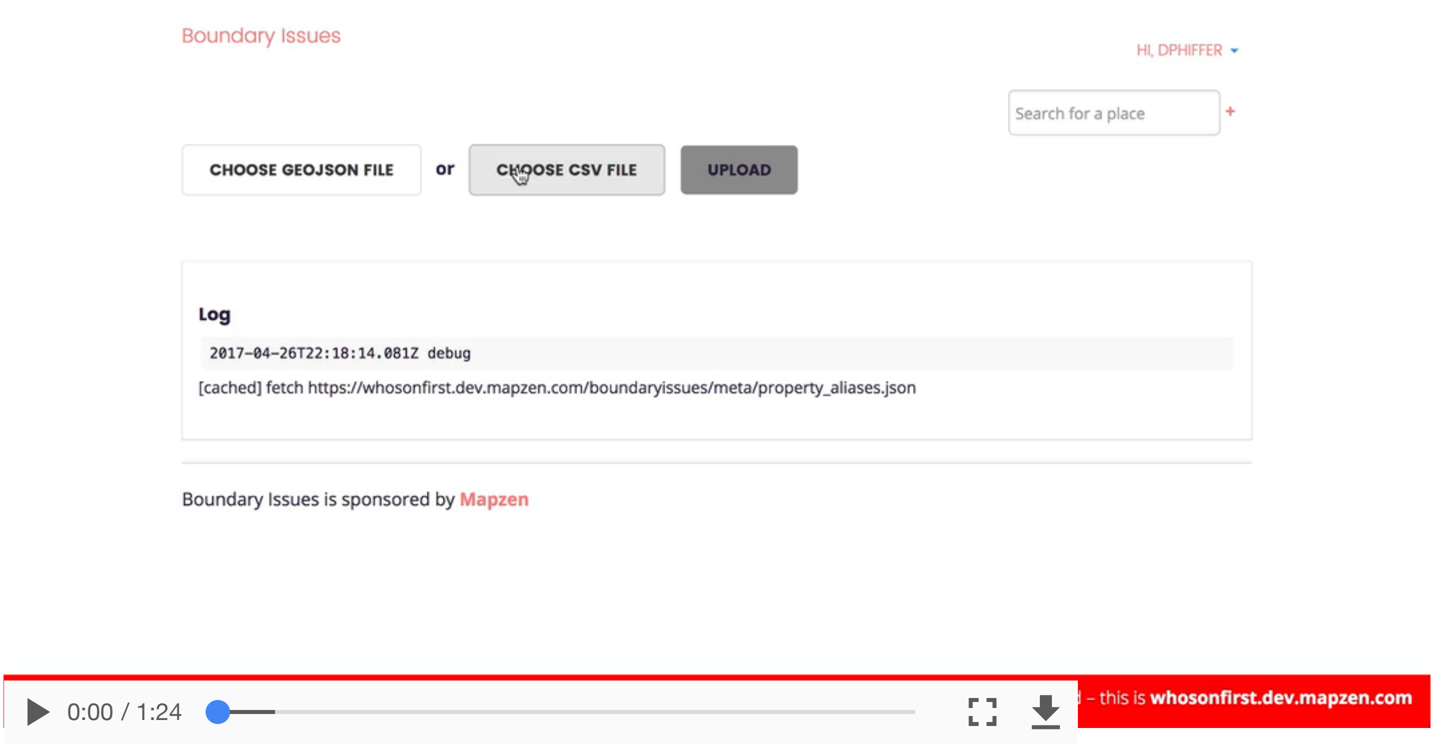
Hello, Transitland!

Both my blog post and talk focus on New York City's subway. As I was thinking about the system, and the 36th St station I used for my example, I realized that we did not actually *have* records for subway stations in Who's On First.

That seemed like as good excuse as any to *import the subway stations*. After all, I just made this fancy new CSV importer, why not put it to use?

So I brought in the records using the **Transitland Datastore API (https://transit.land/documentation/datastore/api-endpoints.html)**. I wrote **a shell script (https://gist.github.com/dphiffer/00aa7b646cd206e067feee1bf7c5b04c)** that generated **a CSV file (https://gist.github.com/dphiffer/00aa7b646cd206e067feee1bf7c5b04c#file-2_transitland_nyc_subway-csv)**, encoding each station as a row. The columns include a station name, a **Onestop ID (https://transit.land/documentation/onestop-id-scheme/)**, and latitude/longitude coordinates.

Here is a screen capture I took while I was testing to give you a sense for the process.


Boundary Issues
HI, DPHIFFER
Search for a place
CHOOSE GEOJSON FILE
or
CHOOSE CSV FILE
UPLOAD
Log
Boundary Issues is sponsored by Mapzen
0:00 / 1:24
– this is whosonfirst.dev.mapzen.com

```
2017-04-26T22:18:14.081Z debug
[cached] fetch https://whosonfirst.dev.mapzen.com/boundaryissues/meta/property_aliases.json
```

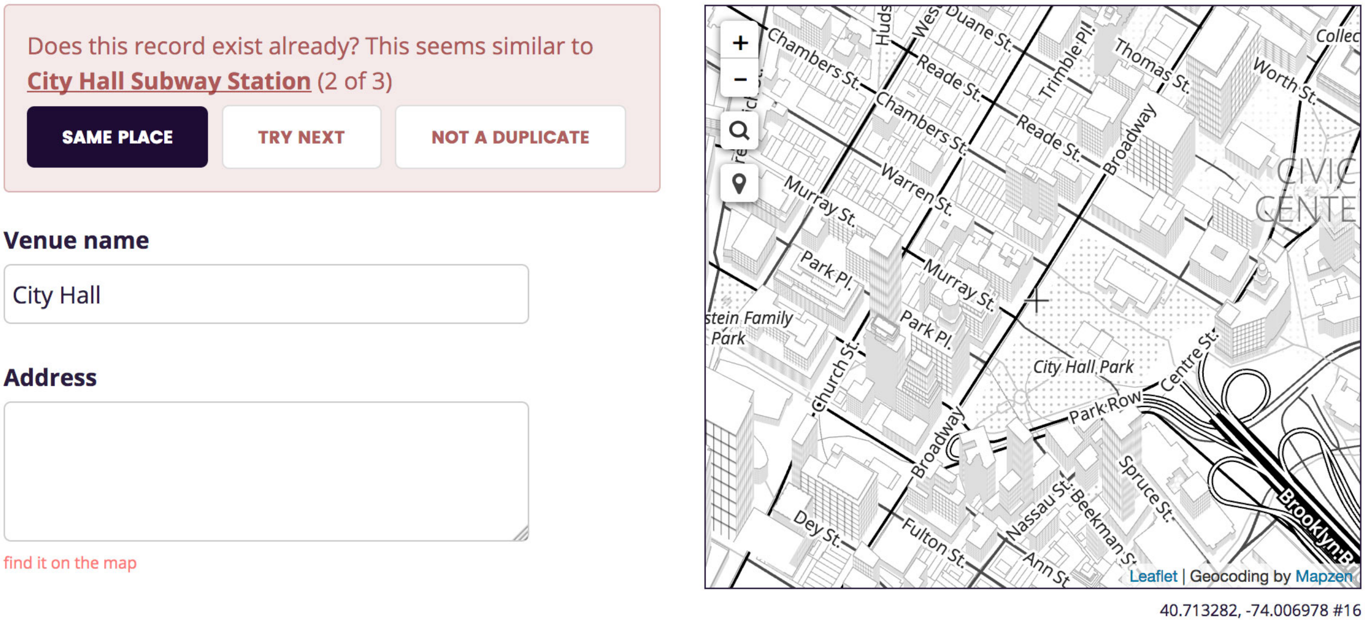
It worked! I imported all of the subway stations, using a home team geometry source. There is one remaining mystery regarding the number of stations; of the 502 CSV rows I discovered a handful of duplicates on the way in, ultimately resulting in **471 records (https://whosonfirst.mapzen.com/spelunker/concordances/transitland/)**. However, according to **Wikipedia (https://en.wikipedia.org/wiki/New_York_City_Subway_stations)** we should actually have 472 stations. In due time we will untangle the station count discrepancy. Better than yesterday!

Don’t get duped by dupes

For each new venue you import during the step-through process there’s a possibility that we already have a WOF record in the dataset. We don’t want two records for the same place, so it’s incumbent upon the user interface to alert you when something with the same name and location exists.

De-duping places is a *hard problem (a.k.a. tricky business)* (***Deduplicating a Places Database (https://research.fb.com/publications/deduplicating-a-places-database/)*** is a paper from Facebook Research that has been recommended to me, although I haven’t read it yet) and I did not set out to solve in a complete or even correct way. What I did set out to do is make a de-duping interface we could live with today and that we could improve on tomorrow. I created the *simplest dumbest thing*, a stepping stone to a *marginally less dumb, more nuanced thing*.

Here’s what it looks like:



Does this record exist already? This seems similar to **City Hall Subway Station** (2 of 3)

Venue name

City Hall

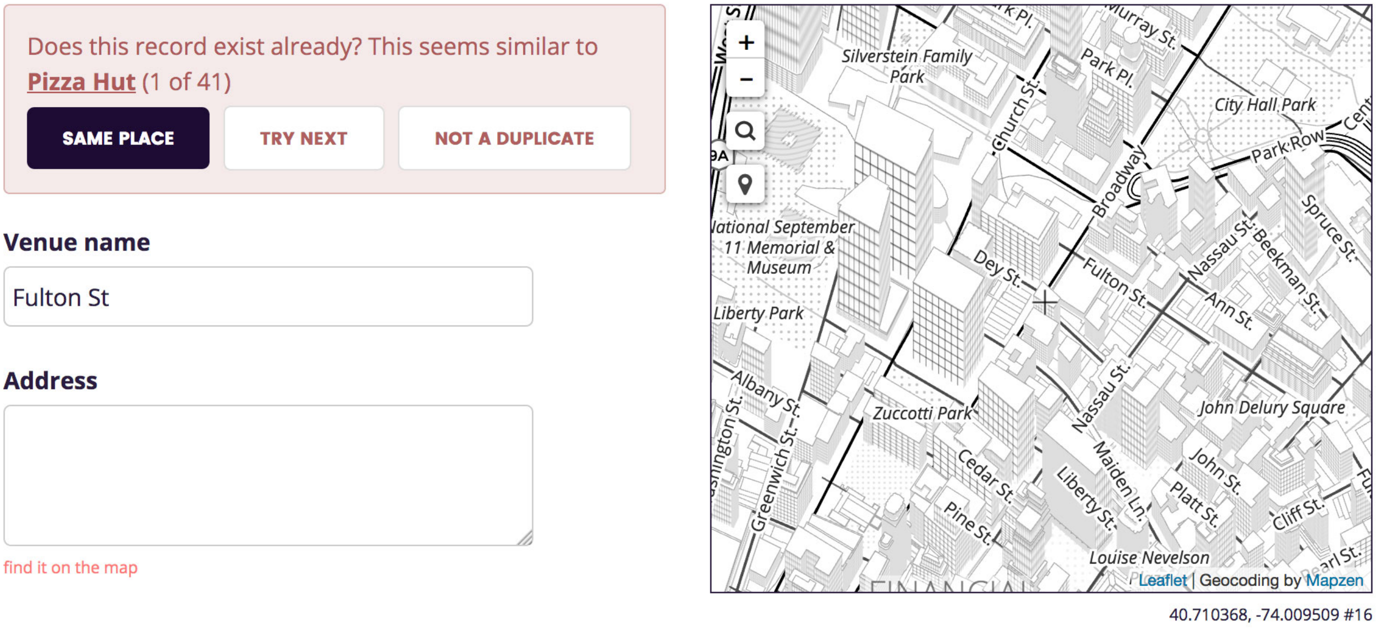
Address

find it on the map

40.713282, -74.006978 #16

SAME PLACE
TRY NEXT
NOT A DUPLICATE
+
−
Chambers St.
Duane St.
Reade St.
Trimble Pl.
Thomas St.
Worth St.
Broadway
Warren St.
Murray St.
Park Pl.
CIVIC CENTE
stein Family Park
Church St.
City Hall Park
Centre St.
Park Row
Spruce St.
Nassau St.
Beekman St.
Dey St.
Fulton St.
Ann St
Brooklyn B
Leaflet | Geocoding by Mapzen

Sometimes it found *obviously wrong* dupe candidates:



Does this record exist already? This seems similar to **Pizza Hut** (1 of 41)

Venue name

Fulton St

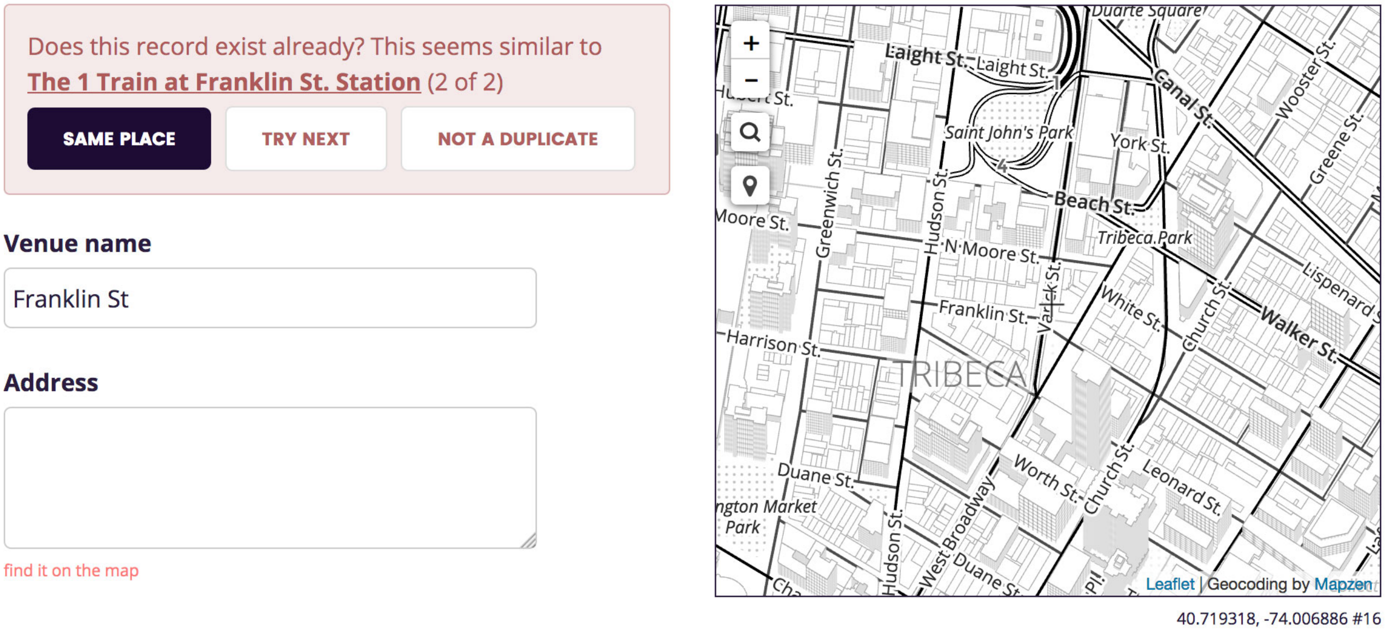
Address

find it on the map

40.710368, -74.009509 #16

SAME PLACE
TRY NEXT
NOT A DUPLICATE
+
−
Silverstein Family Park
Murray St.
Park Pl.
Church St.
City Hall Park
Park Row
Broadway
National September 11 Memorial & Museum
Liberty Park
Dey St.
Fulton St.
Nassau St.
Beekman St.
Spruce St.
Ann St.
Albany St.
Zuccotti Park
John Delury Square
Greenwich St.
Cedar St.
Liberty St.
Maiden Ln.
John St.
Platt St.
Pine St.
Cliff St.
Louise Nevelson
FINANCIAL
Leaflet | Geocoding by Mapzen

And sometimes it surfaced incorrect (or at least *incomplete*) WOF records:



Does this record exist already? This seems similar to **The 1 Train at Franklin St. Station** (2 of 2)

SAME PLACE
TRY NEXT
NOT A DUPLICATE

Venue name

Franklin St

Address

find it on the map

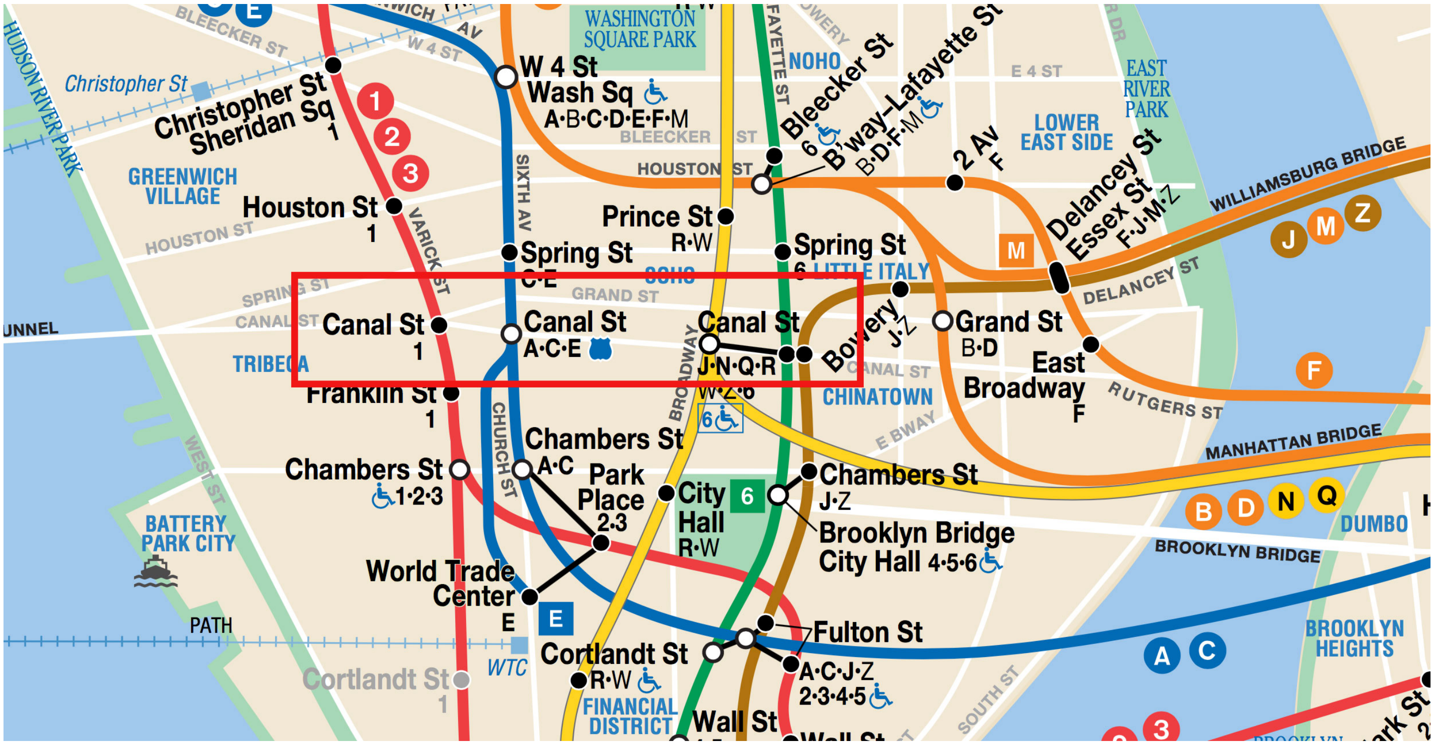
+
−
Duarte Square
Laight St.
Laight St.
Canal St.
Wooster St.
Greene St.
Saint John's Park
York St.
Beach St.
Greenwich St.
Hudson St.
Moore St.
N Moore St.
Tribeca Park
Lispenard St.
Franklin St.
Varick St.
White St.
Church St.
Walker St.
Harrison St.
TRIBECA
Duane St.
Worth St.
Leonard St.
Church St.
Broadway
West Broadway
Duane St.
Hudson St.
Washington Market Park
Leaflet | Geocoding by Mapzen
40.719318, -74.006886 #16

Despite the mixed quality of the dupe candidates, it worked! I was able to discover and merge the new data with existing WOF records.

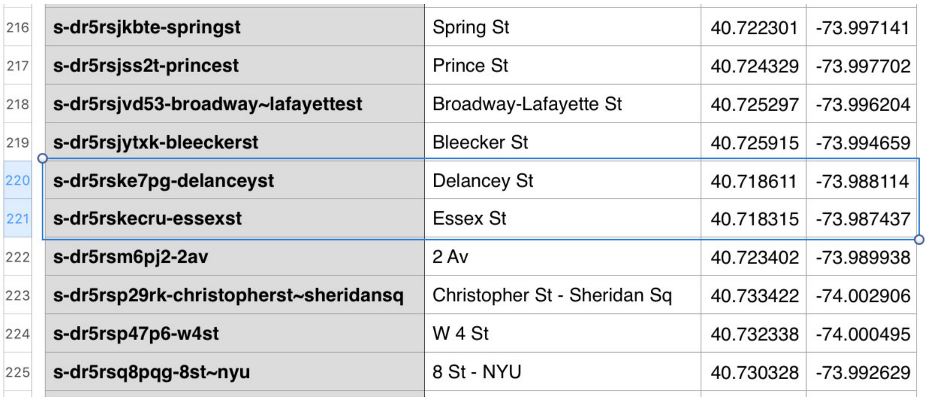
One caveat to add is that my dupe-detection UI has both false positives (suggestions for places that aren't the same) and false negatives (it fails to suggest some existing places). I am more concerned with fixing the latter, but in the interim I have added a method for manually specifying a duplicate place by WOF ID.

The Canal St Station problem

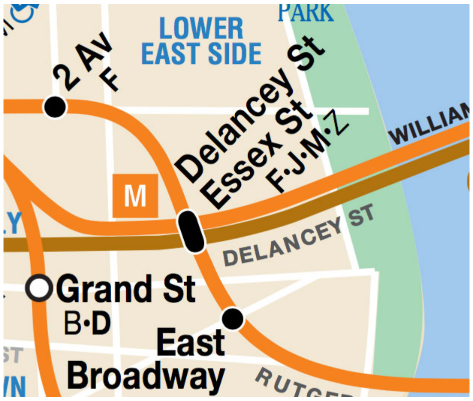
I came to learn quickly that NYC's subway stations are full of challenging edge cases. Lots of stations have the same name! Canal Street has five different subway stations, from which you can take the 1, A, C, E, J, N, Q, R, J, or Z trains.


Christopher St
Christopher St Sheridan Sq
1
W 4 St Wash Sq
A·B·C·D·E·F·M
Bleecker St
6
B'way-Lafayette St
B·D·F·M
2 Av
F
Houston St
1
Prince St
R·W
Spring St
C·E
Spring St
6
Delancey St Essex St
F·J·M·Z
Canal St
1
Canal St
A·C·E
Canal St
J·N·Q·R
W·Z·6
Bowery
J·Z
Grand St
B·D
East Broadway
F
Franklin St
1
Chambers St
A·C
Chambers St
1·2·3
Park Place
2·3
City Hall
R·W
Chambers St
J·Z
Brooklyn Bridge City Hall 4·5·6
World Trade Center
E
Cortlandt St
R·W
Cortlandt St
1
Fulton St
A·C·J·Z
2·3·4·5
Wall St
WASHINGTON SQUARE PARK
NOHO
GREENWICH VILLAGE
LOWER EAST SIDE
EAST RIVER PARK
SOHO
LITTLE ITALY
TRIBECA
CHINATOWN
BATTERY PARK CITY
FINANCIAL DISTRICT
BROOKLYN HEIGHTS
DUMBO
WILLIAMSBURG BRIDGE
MANHATTAN BRIDGE
BROOKLYN BRIDGE
HUDSON RIVER PARK
PATH
WTC

As the data was coming in, I had to determine if the incoming record actually represented a *new Canal St* or another dimension of an existing record. To make matters more confusing, Transitland accounts for each platform within a subway station complex.



216	**s-dr5rsjkbte-springst**	Spring St	40.722301	-73.997141
217	**s-dr5rsjss2t-princest**	Prince St	40.724329	-73.997702
218	**s-dr5rsjvd53-broadway~lafayettest**	Broadway-Lafayette St	40.725297	-73.996204
219	**s-dr5rsjytxk-bleeckerst**	Bleecker St	40.725915	-73.994659
220	**s-dr5rske7pg-delanceyst**	Delancey St	40.718611	-73.988114
221	**s-dr5rskecru-essexst**	Essex St	40.718315	-73.987437
222	**s-dr5rsm6pj2-2av**	2 Av	40.723402	-73.989938
223	**s-dr5rsp29rk-christopherst~sheridansq**	Christopher St - Sheridan Sq	40.733422	-74.002906
224	**s-dr5rsp47p6-w4st**	W 4 St	40.732338	-74.000495
225	**s-dr5rsq8pqg-8st~nyu**	8 St - NYU	40.730328	-73.992629


2 Av
F
LOWER EAST SIDE
PARK
Delancey St Essex St
F·J·M·Z
M
WILLIAM
DELANCEY ST
Grand St
B·D
East Broadway
RUTGE

I decided on a rule that I could follow: if there's a *dot on the official MTA map* that dot represents a distinct station. Even if you can walk between two of the dots on the map via a connecting tunnel—DOT EQUALS STATION. Anything more complicated elevates the likelihood of forming the wrong mental model for the next person to try to understand.

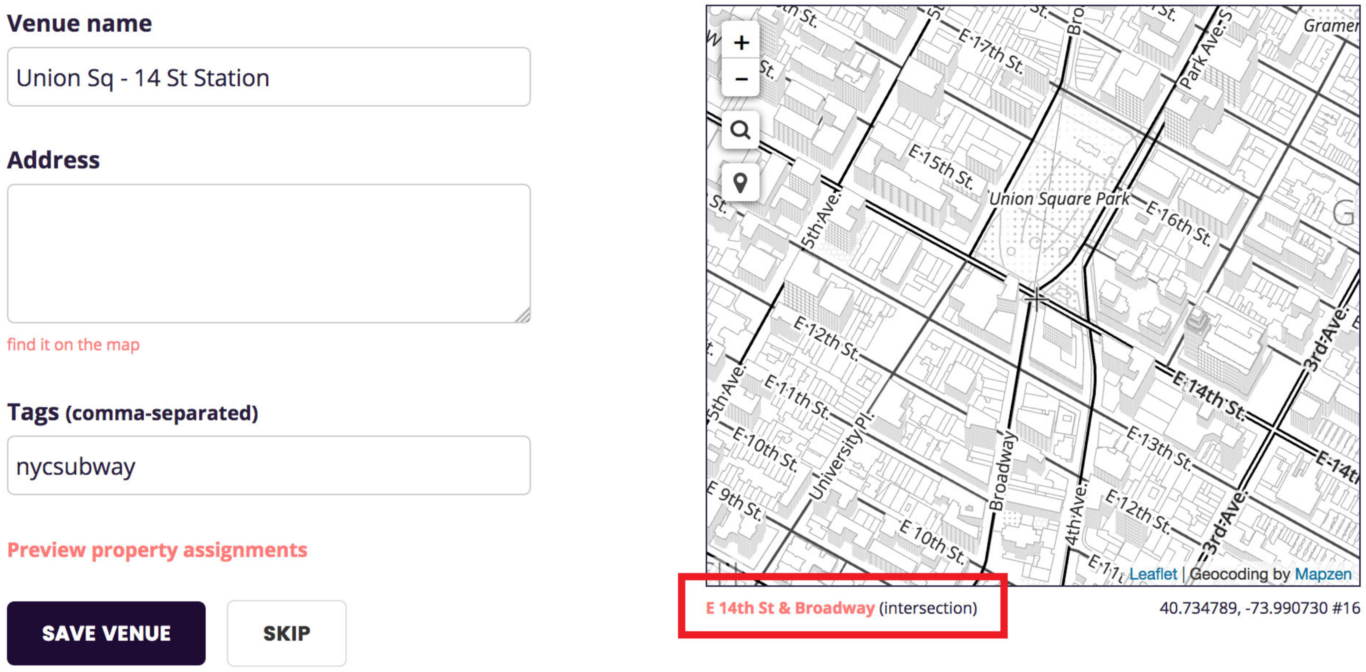
As I was writing this blog post, I heard about some new Transitland updates that are in the works. **Ian Rees (https://github.com/irees)** along with **Greg (https://twitter.com/heyknisely)** and **Duane (https://twitter.com/DuaneGearhart)** have been working on Transitland's station hierarchies, which capture station/platform/entrance relationships. More is coming soon!

Place hierarchies are hard

For each record we add to Who’s On First, we wire in its hierarchy of parent places. For example, **14 St - Union Square Station (https://whosonfirst.mapzen.com/spelunker/id/1108958717/)** is in Union Square neighbourhood, in Manhattan, which is in New York City, etc. Additionally, each place has a direct `wof:parent_id` assigned to it.

For some reason during my import process the hierarchies came out all funky. Some, but not all, records were parented by `borough` records (skipping `intersection`, `microhood`, `neighbourhoud`, etc.). Other records had no hierarchy or parent. Unfortunately my *very pared down, very simple* user interface didn’t provide any indication of the hierarchy (or lack thereof).

I have since added a small indicator that shows what the direct parent of the place is, which should help clue in the person importing data when something goes wrong.



Venue name

Union Sq - 14 St Station

Address

find it on the map

Tags (comma-separated)

nycsubway

Preview property assignments

E 14th St & Broadway (intersection)

40.734789, -73.990730 #16

SAVE VENUE
SKIP
E 17th St.
E 15th St.
Union Square Park
E 16th St.
5th Ave.
Park Ave. S
E 12th St.
E 11th St.
E 10th St.
E 9th St.
University Pl.
Broadway
4th Ave.
E 14th St.
E 13th St.
3rd Ave.
Leaflet | Geocoding by Mapzen

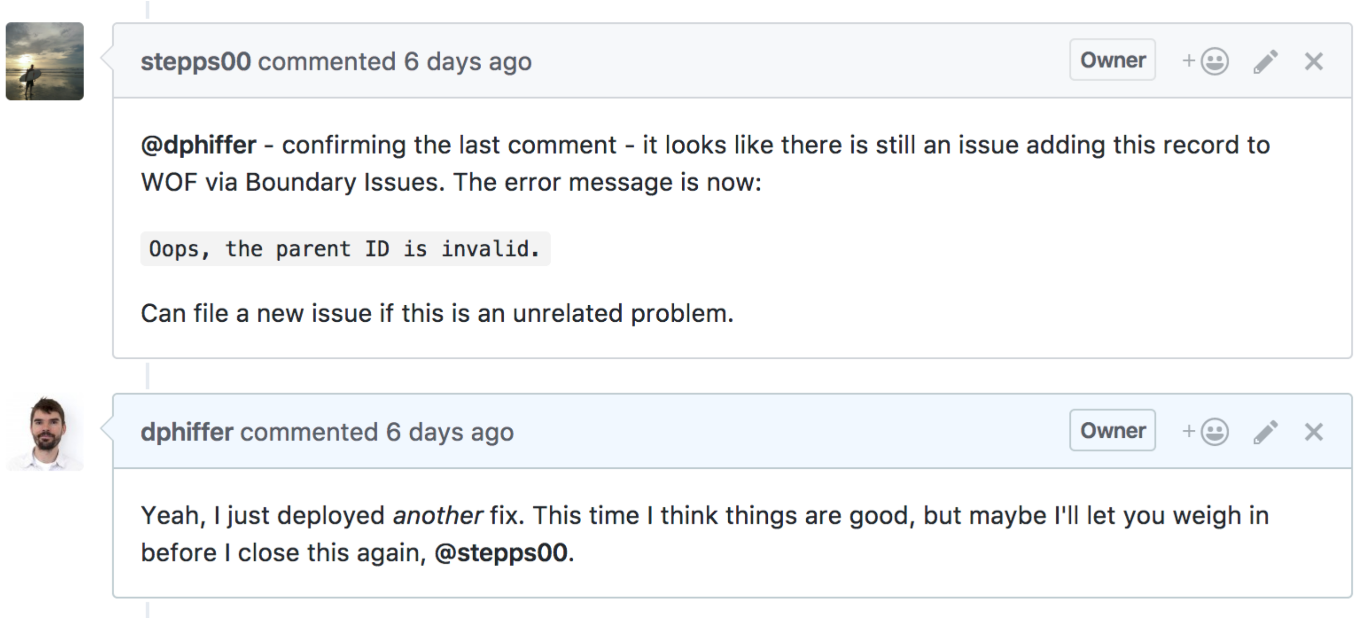
For a more in-depth description of the Who’s On First placetype hierarchy, see the **whosonfirst-placetype repo (https://github.com/whosonfirst/whosonfirst-placetypes)**.

Shipping good bug fixes is hard

Once I discovered the importer was generating bad hierarchies and assigning the wrong parent IDs, I decided to add a check that would do some validation before the record gets saved.

I deployed the validation check, Aaron helped me clean up the incorrectly-encoded subway records, and then I went back to enjoying the csv,conf presentations.

Almost immediately I heard from three people, more or less in unison, that something was wrong. They could not save WOF records using the importer! Something about an `Incorrect parent ID`.



stepps00 commented 6 days ago

@dphiffer - confirming the last comment - it looks like there is still an issue adding this record to WOF via Boundary Issues. The error message is now:

`Oops, the parent ID is invalid.`

Can file a new issue if this is an unrelated problem.

dphiffer commented 6 days ago

Yeah, I just deployed *another* fix. This time I think things are good, but maybe I'll let you weigh in before I close this again, **@stepps00**.

So I went back to the drawing board. There turned out to be a flaw in how we were requesting **point-in-polygon (https://en.wikipedia.org/wiki/Point_in_polygon)** results. However, the larger problem was that overzealous validation was blocking the save process. A person adding a venue just wants the *save button* to *save the thing*. They don't know what a parent ID error is, especially when they have no means of fixing it.

I'm not sure whether I've nailed down all the root causes for bad `wof:hierarchy` and `wof:parent_id` properties, but I have come up with a better way to mitigate that situation if it crops up again. Now when you press the save button, the code does a slightly looser validation check, and if the hierarchy/parent stuff looks wrong—it will tell you two things:

1. Hey, the hierarchy/parent stuff looks bad
2. But don't worry I'm fixing it (try saving again!)

And then it goes and fixes it. For now you still have to press the save button one more time, which is a temporary compromise. On the one hand, I am curious to know how often people encounter this particular problem. On the other hand, you just want to *save the thing*, and *why can't it just fix-and-then-save automatically?* Today it is *somewhat* annoying, but not to the degree that it stops you from doing what you're trying to do.

Still to come

All this time I was fixing issues with the CSV importer, I was testing out another subway data set I got from the official **NYC Open Data website (https://data.cityofnewyork.us/Transportation/Subway-Stations/arq3-7z49)**.

It's the same list of subway stations, but it includes some interesting summarized descriptions of what you can find at each station in terms of train lines and service frequency.

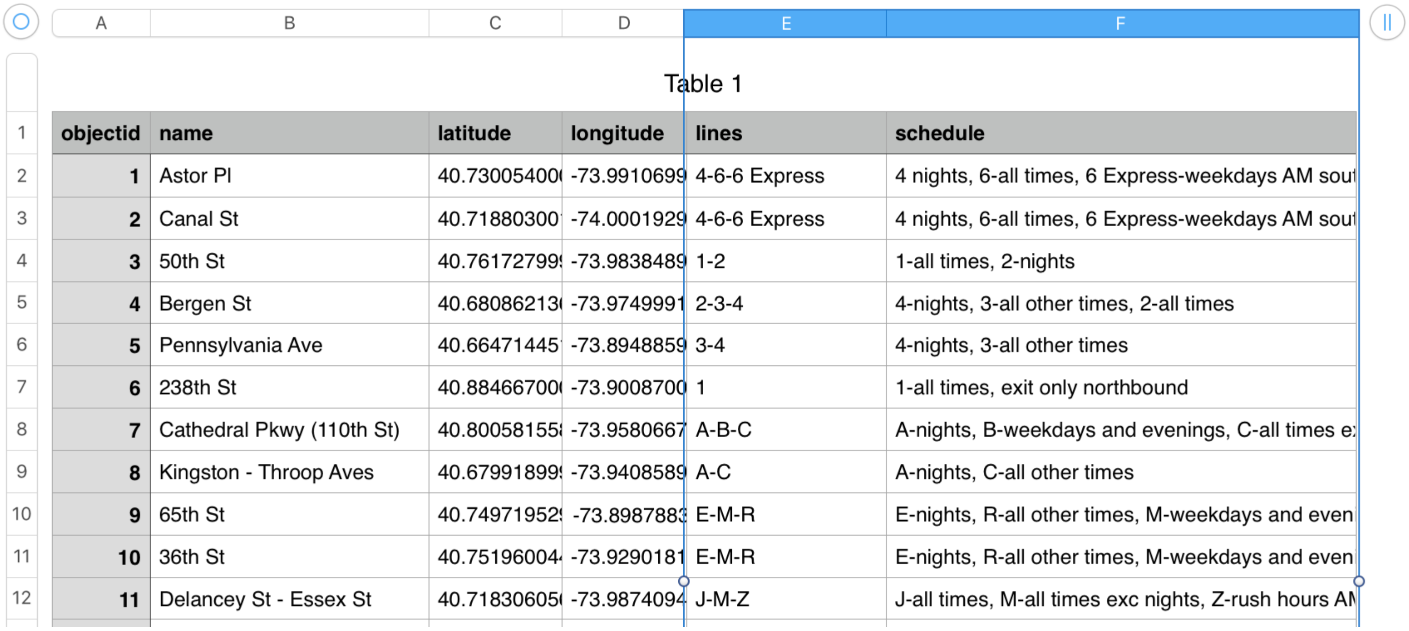
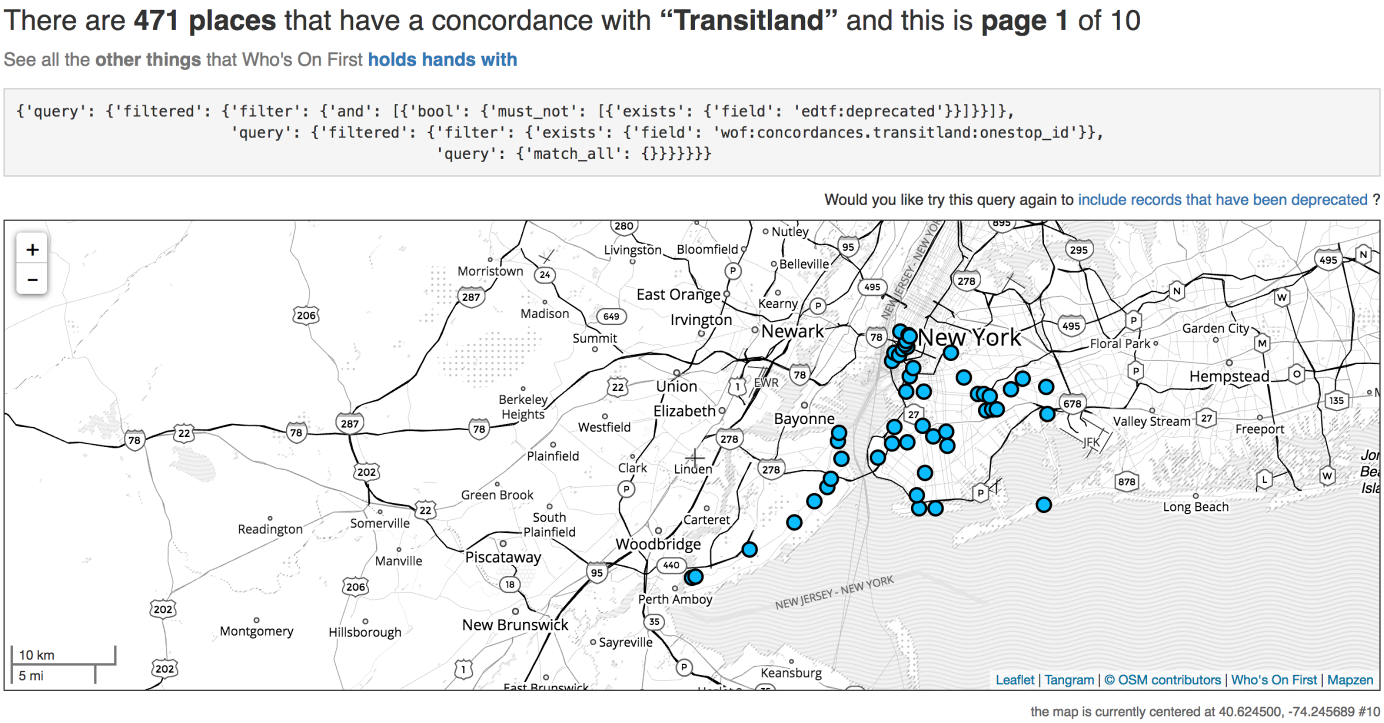


Table 1

objectid	name	latitude	longitude	lines	schedule
1	Astor Pl	40.73005400	-73.9910699	4-6-6 Express	4 nights, 6-all times, 6 Express-weekdays AM sou
2	Canal St	40.71880300	-74.0001929	4-6-6 Express	4 nights, 6-all times, 6 Express-weekdays AM sou
3	50th St	40.76172799	-73.9838489	1-2	1-all times, 2-nights
4	Bergen St	40.68086213	-73.9749991	2-3-4	4-nights, 3-all other times, 2-all times
5	Pennsylvania Ave	40.66471445	-73.8948859	3-4	4-nights, 3-all other times
6	238th St	40.88466700	-73.9008700	1	1-all times, exit only northbound
7	Cathedral Pkwy (110th St)	40.80058155	-73.9580667	A-B-C	A-nights, B-weekdays and evenings, C-all times e
8	Kingston - Throop Aves	40.67991899	-73.9408589	A-C	A-nights, C-all other times
9	65th St	40.74971952	-73.8987883	E-M-R	E-nights, R-all other times, M-weekdays and even
10	36th St	40.75196004	-73.9290181	E-M-R	E-nights, R-all other times, M-weekdays and even
11	Delancey St - Essex St	40.71830605	-73.9874094	J-M-Z	J-all times, M-all times exc nights, Z-rush hours A

More importantly, it means we will have records with both the official object ID assigned from the NYC Open Data website *and also* the concordance from Transitland. If any other application uses the "official" data source, they can now use a WOF record to correlate *their stuff* with *Transitland's stuff*.

Until then, here are the **Transitland concordances (https://whosonfirst.mapzen.com/spelunker/concordances/transitland/)** we do have.



There are **471 places** that have a concordance with **“Transitland”** and this is **page 1** of 10

See all the **other things** that Who's On First **holds hands with**

```
{'query': {'filtered': {'filter': {'and': [{'bool': {'must_not': [{'exists': {'field': 'edtf:deprecated'}}]}}]},
                        'query': {'filtered': {'filter': {'exists': {'field': 'wof:concordances.transitland:onestop_id'}},
                                               'query': {'match_all': {}}}}}}}
```

Would you like try this query again to include records that have been deprecated ?

the map is currently centered at 40.624500, -74.245689 #10

+
−
New York
Newark
Morristown
Livingston
Bloomfield
Nutley
Belleville
East Orange
Irvington
Kearny
Madison
Summit
Union
Elizabeth
Bayonne
Berkeley Heights
Westfield
Plainfield
Clark
Linden
Green Brook
South Plainfield
Somerville
Readington
Manville
Piscataway
Carteret
Woodbridge
Perth Amboy
New Brunswick
Sayreville
Keansburg
Montgomery
Hillsborough
East Brunswick
Floral Park
Garden City
Hempstead
Valley Stream
Freeport
Long Beach
JFK
EWR
NEW JERSEY - NEW YORK
10 km
5 mi
Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen

It just *looks* simple

A big motivation for creating a CSV importer has been to reduce the complexity involved in getting data into Who’s On First. I created a venue editor with just three text inputs: *name*, *address*, and *tags*, instead of the myriad property editing controls of the fully featured edit page. Without having studied the full corpus of **WOF theory (https://whosonfirst.mapzen.com/theory/)**, someone with a CSV file to share should be able to contribute to the dataset.

Enter a name, pick a location, hit the save button. You put the data in, and *boing*, your place is on the map. (Shout out to my mom, Kathleen Phiffer, whose one sentence-summary of my talk I borrowed here.)

My simpler add-a-venue page *still produces fully-functional GeoJSON*. There’s a whole lot going on behind the scenes that’s easy to miss.

Do you have good open data you want to try? **Drop us a line (mailto:boundaryissues@mapzen.com)** if you want to test.

Excuse me while I promote one more thing

Plug: *I'm going to be giving a shorter version of my csv,conf talk at month's GeoNYC. It is happening on Monday May 15, so if you don't have plans already come on out to **Mapzen (https://whosonfirst.mapzen.com/spelunker/id/420573371/)** to hear all about how **POIs are just interesting points (https://www.meetup.com/geonyc/events/239820070/)**.*

· 10 May 2017 ·

Dan Phiffer

Dan is a data carpenter, which tbh is kind of a made up title. But he likes it all the same.

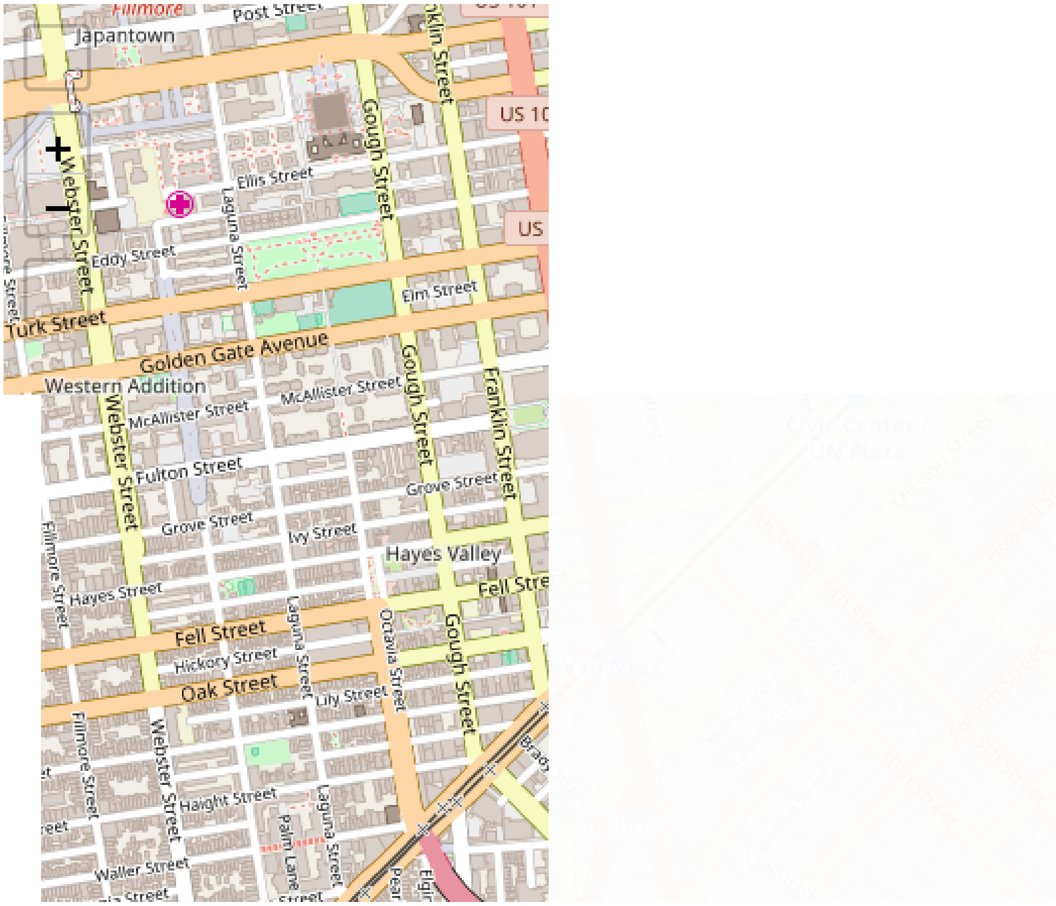
© 2017 Mapzen

Bike to Work with Mapzen's updated bike map

cartography (/tag/cartography) tangram (/tag/tangram) vector-tiles (/tag/vector-tiles)

Since we **launched our bike map in March (https://mapzen.com/blog/bike-map/)**, we've gotten lots of great feedback. Nearly a hundred of you took our survey, and we've been listening! Thank you! We've learned lots about biking experiences all over the world, how bike infrastructure varies from city to city, and the many ways people like to bike, along with the comfort levels of of what cyclists consider safe or unsafe.

Today we are releasing version 2 of the Walkabout bike map. This new iteration adjusts the color scheme, makes safer routes more prominent on the map, and combines bike route relations with bike infrastructure and street types. Happy **Bike to Work Day (https://youcanbikethere.com/2017/05/get-ready-to-bike-to-work-on-may-11/)** and **Month (http://bikeleague.org/bikemonth)**! 🚲

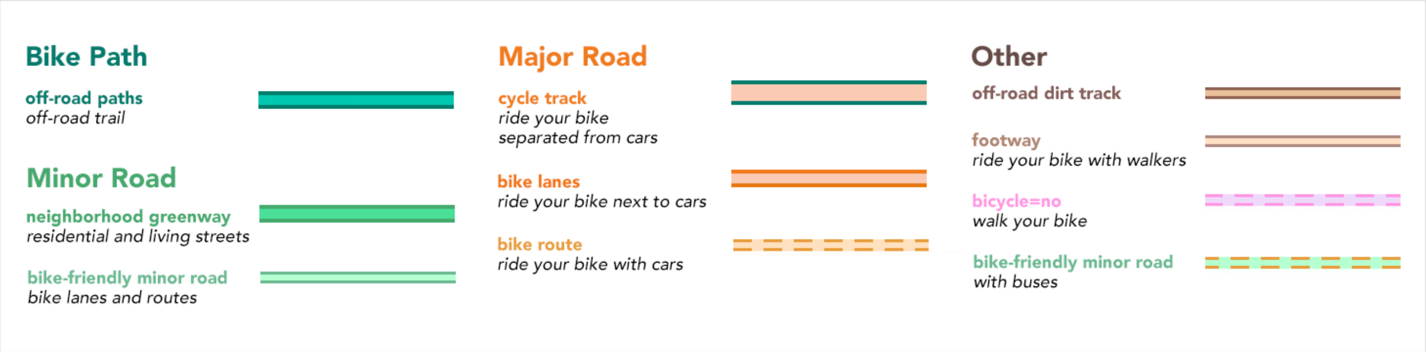

Japantown
Post Street
Franklin Street
US 101
Gough Street
Ellis Street
Laguna Street
Webster Street
Eddy Street
Elm Street
Turk Street
Golden Gate Avenue
Western Addition
McAllister Street
Fulton Street
Grove Street
Ivy Street
Hayes Valley
Fell Street
Fillmore Street
Hayes Street
Hickory Street
Oak Street
Octavia Street
Lily Street
Haight Street
Waller Street
Palm Lane
Civic Center / UN Plaza
Van Ness

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

*(This map is interactive! **Open full screen ➶ (https://mapzen.com/products/maps/walkabout/bike-map/)**)*

Legend: take two

No more blue! More green, plus dashed orange.


Bike Path

off-road paths
off-road trail

Minor Road

neighborhood greenway
residential and living streets

bike-friendly minor road
bike lanes and routes

Major Road

cycle track
ride your bike separated from cars

bike lanes
ride your bike next to cars

bike route
ride your bike with cars

Other

off-road dirt track

footway
ride your bike with walkers

bicycle=no
walk your bike

bike-friendly minor road
with buses

As usual, you can enable the bike overlay **in just a few lines**!

We know that people have different comfort levels and thresholds for safety when cycling. Some people want separated lanes (dark green), while others are OK riding alongside cars (orange lines), while yet others want to avoid buses (dashed orange edges and green fill).

Our goal is to give you enough information so you can make an informed decision on what's safe for you, whether you're biking near home or in places you might not be very familiar with.

In redesigning the bike map we kept our earlier questions in mind:

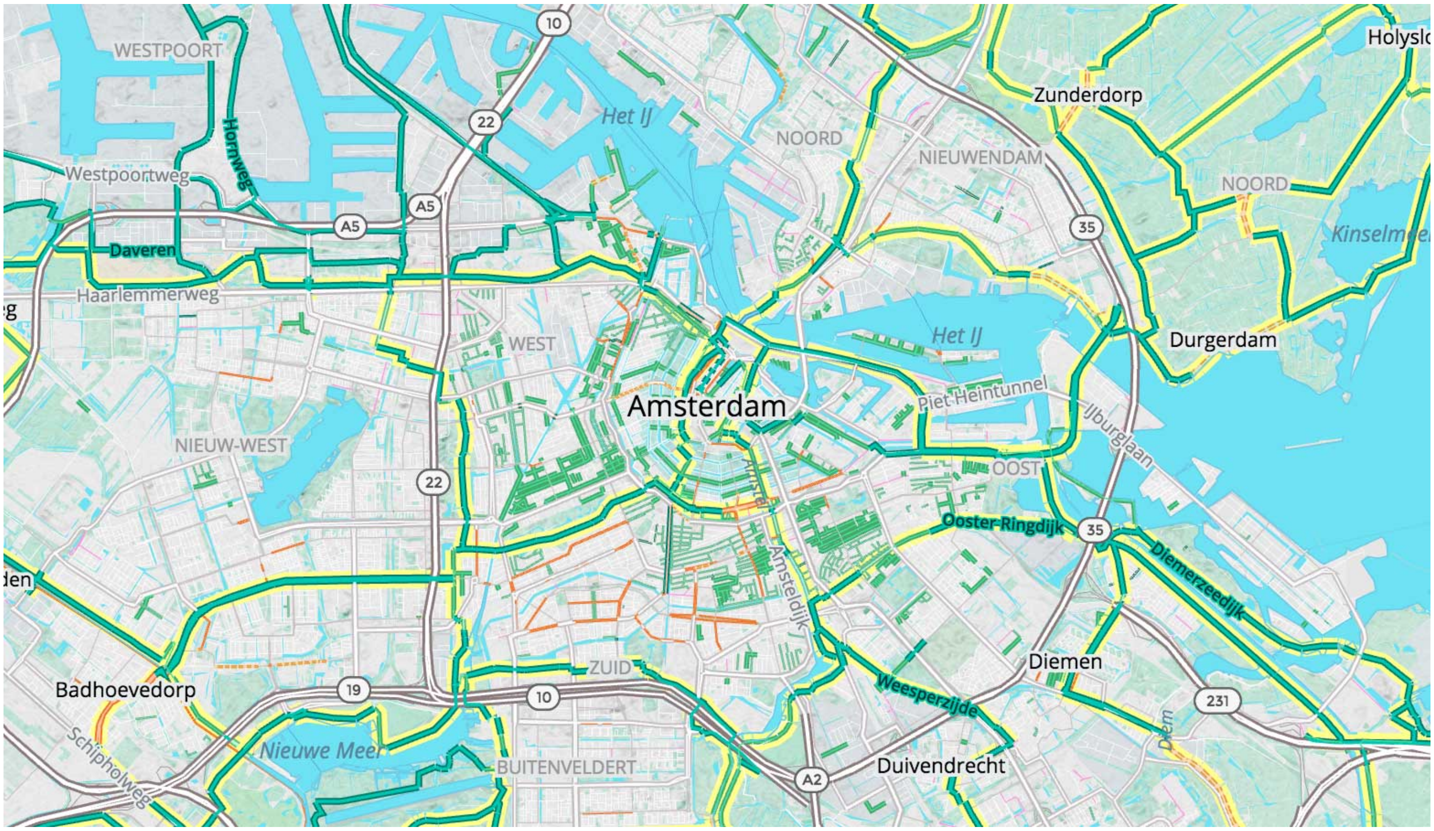
- What is the safest route?
- Do I have to ride with (lots of) cars (or buses)?
- Which side of the road should I ride on?

Take a look at the gallery, or **jump below** to learn how we took into account both road type, bicycle network, and bicycle infrastructure to emphasize safer routes on our updated bike map.

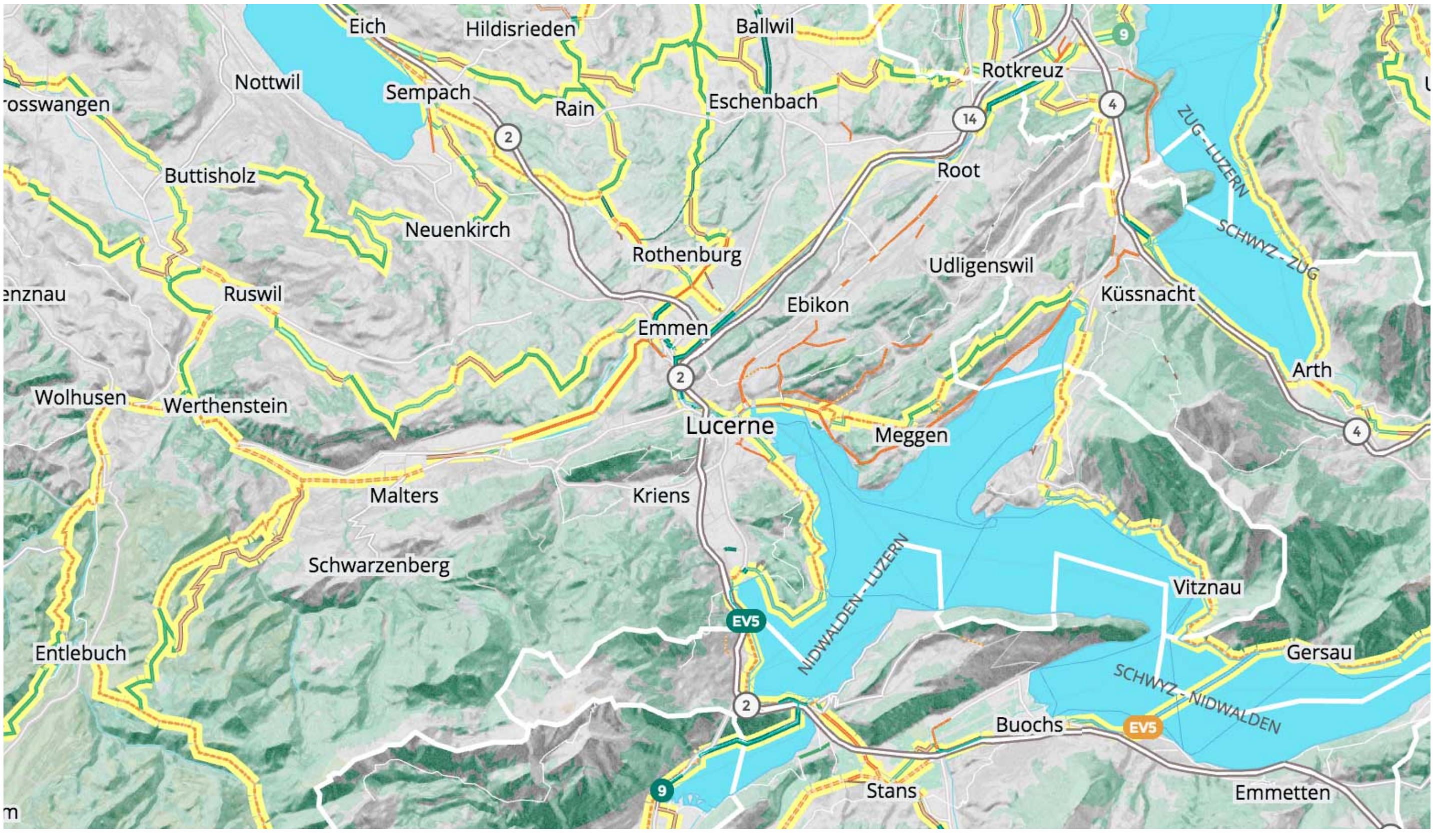
We'd love to hear your feedback **in our poll (https://docs.google.com/forms/d/1pAi1hWj8tW7ssxo9lOjW1xtcRYo3RUswvVQSmTxbkqg)**!

Gallery

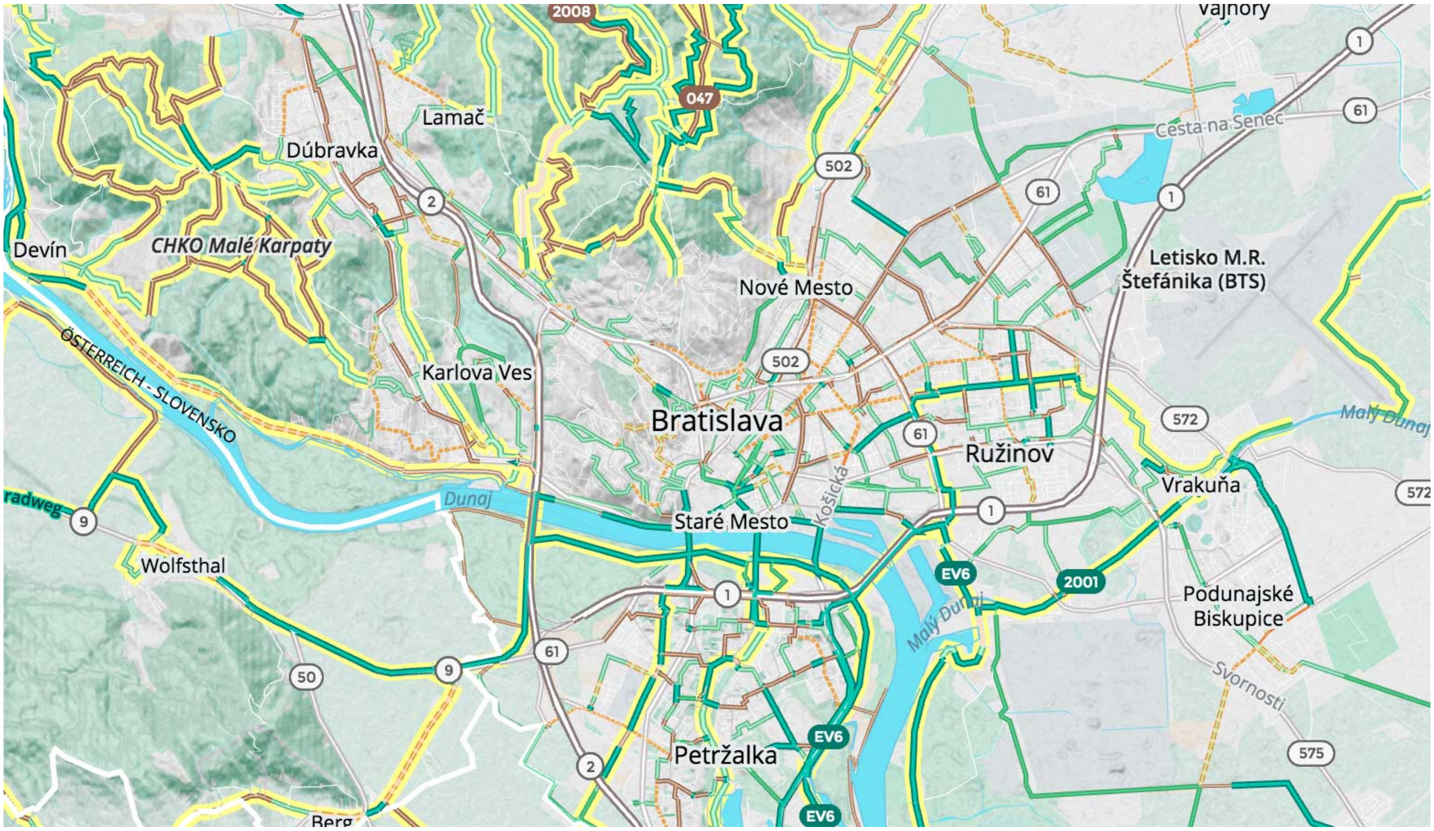
Thanks again to everyone who responded to our **poll (https://docs.google.com/forms/d/1pAi1hWj8tW7ssxo9lOjW1xtcRYo3RUswvVQSmTxbkqg)** (which is still open!) Here are some previews for you.


WESTPOORT
10
22
Het IJ
NOORD
NIEUWENDAM
Zunderdorp
Holysl
Hornweg
Westpoortweg
A5
A5
35
NOORD
Kinselme
Daveren
Haarlemmerweg
WEST
Het IJ
Durgerdam
Amsterdam
Piet Heintunnel
IJburglaan
NIEUW-WEST
22
OOST
Ooster Ringdijk
35
Diemerzeedijk
Amsteldijk
Amstel
ZUID
Diemen
Badhoevedorp
19
10
Weesperzijde
231
Schipholweg
Nieuwe Meer
BUITENVELDERT
A2
Duivendrecht
Diem

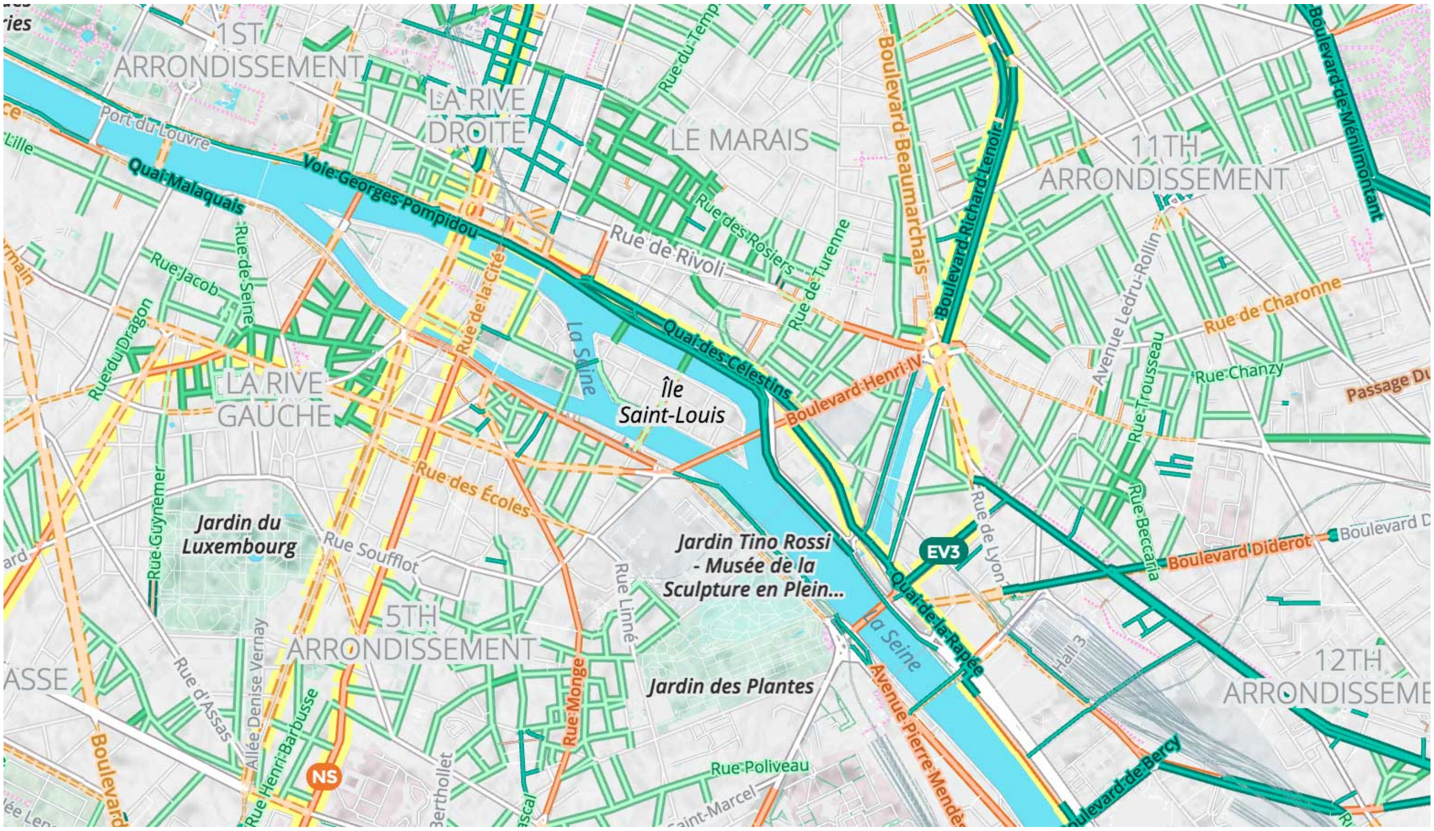
Amsterdam Zoom 12


Eich
Hildisrieden
Ballwil
9
Nottwil
Sempach
Rotkreuz
rosswangen
Rain
Eschenbach
14
4
2
ZUG - LUZERN
Root
Buttisholz
Neuenkirch
SCHWYZ - ZUG
Rothenburg
Udligenswil
enznau
Ruswil
Ebikon
Küssnacht
Emmen
Arth
Wolhusen
Werthenstein
2
Lucerne
Meggen
4
Malters
Kriens
Schwarzenberg
NIDWALDEN - LUZERN
Vitznau
EV5
Entlebuch
Gersau
SCHWYZ - NIDWALDEN
2
Buochs
EV5
9
Stans
Emmetten
m

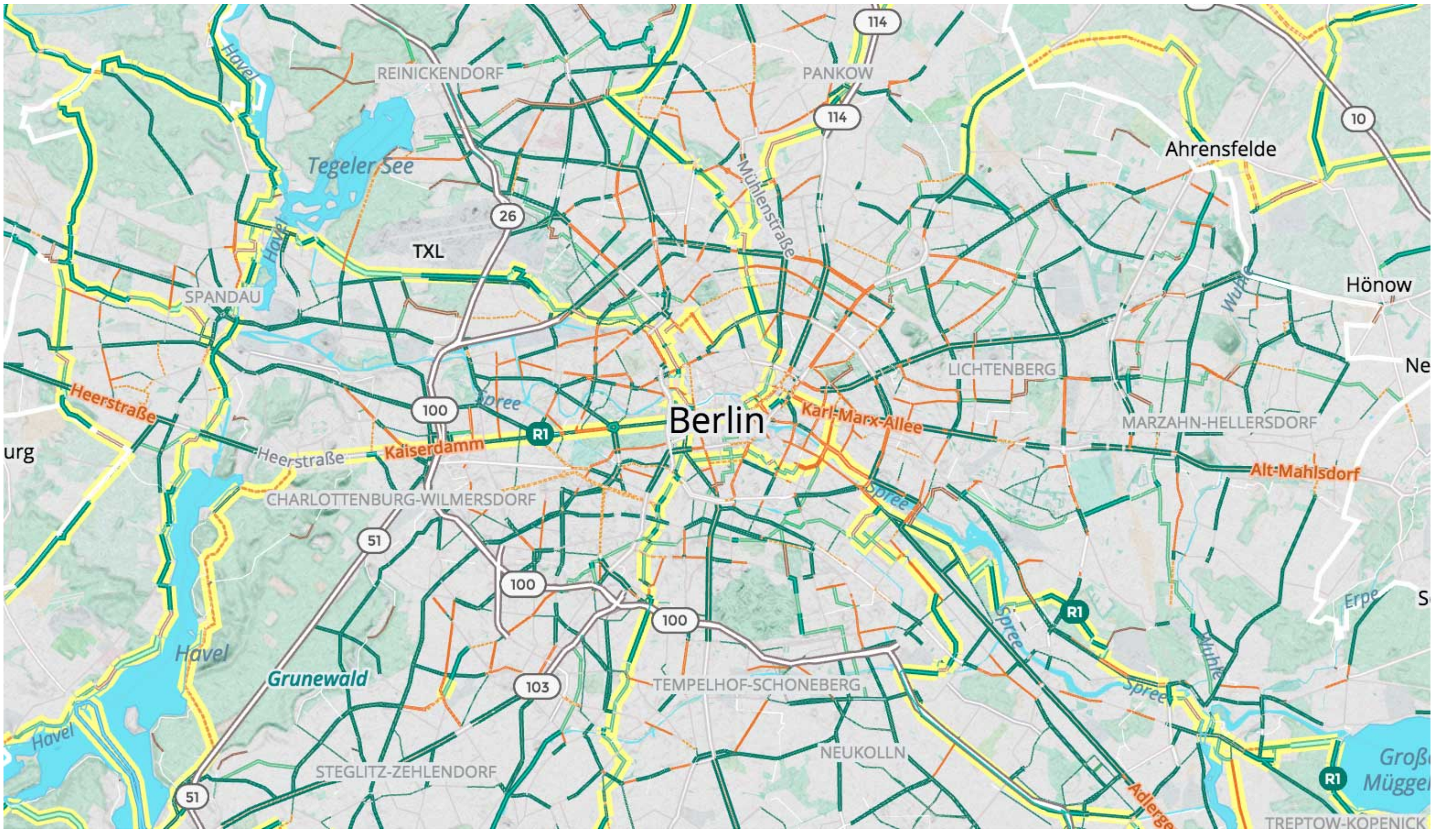
Lucerne Zoom 11


2008
Vajnory
047
Lamač
Dúbravka
Cesta na Senec
502
61
1
2
Devín
CHKO Malé Karpaty
Letisko M.R. Štefánika (BTS)
Nové Mesto
ÖSTERREICH - SLOVENSKO
Karlova Ves
502
Bratislava
572
Malý Dunaj
61
Ružinov
Košická
Vrakuňa
572
radweg
9
Dunaj
Staré Mesto
1
Wolfsthal
EV6
2001
1
Podunajské Biskupice
Malý Dunaj
61
50
9
Svornosti
EV6
575
2
Petržalka
EV6
Berg

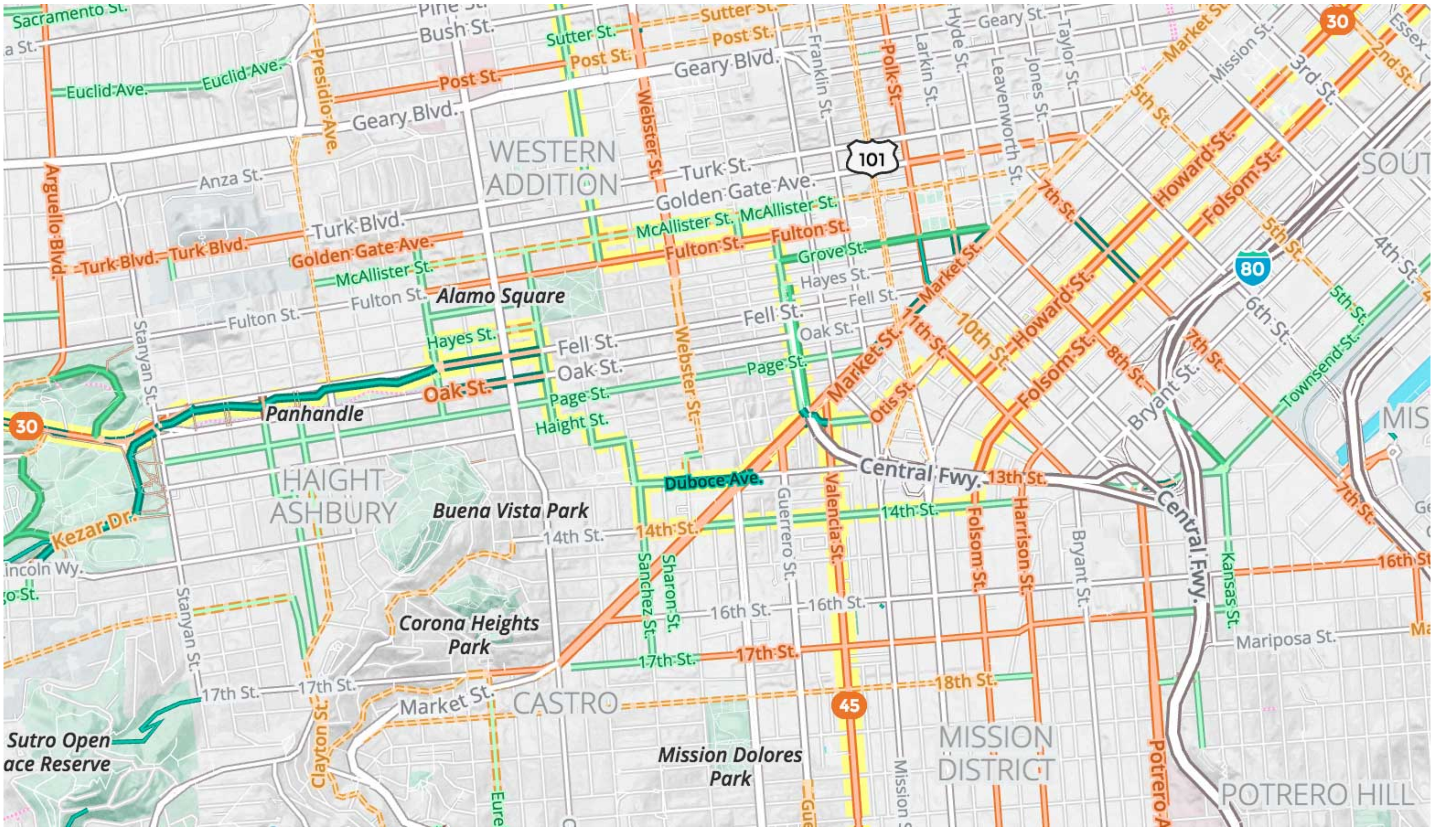
Bratislava Zoom 12


1ST
ARRONDISSEMENT
LA RIVE
DROITE
LE MARAIS
11TH
ARRONDISSEMENT
Port du Louvre
Quai Malaquais
Voie Georges Pompidou
Rue de Rivoli
Rue des Rosiers
Rue de Turenne
Boulevard Beaumarchais
Boulevard Richard Lenoir
Boulevard de Ménilmontant
Avenue Ledru-Rollin
Rue de Charonne
Rue Chanzy
Passage Du
Rue Jacob
Rue de Seine
Rue du Dragon
Rue de la Cité
La Seine
Île
Saint-Louis
Quai des Célestins
Boulevard Henri IV
Rue Trousseau
LA RIVE
GAUCHE
Rue des Écoles
Jardin du
Luxembourg
Rue Soufflot
Rue Guynemer
Jardin Tino Rossi
- Musée de la
Sculpture en Plein...
EV3
Rue de Lyon
Rue Beccaria
Boulevard Diderot
Quai de la Rapée
5TH
ARRONDISSEMENT
Rue Linné
Jardin des Plantes
Rue Monge
Rue d'Assas
Allée Denise Vernay
Rue Henri Barbusse
NS
Rue Poliveau
Avenue Pierre Mendès
12TH
ARRONDISSEME
Boulevard de Bercy
Hall 3

Paris Zoom 14


114
REINICKENDORF
PANKOW
114
Tegeler See
26
TXL
Mühlenstraße
Ahrensfelde
10
Hönow
SPANDAU
LICHTENBERG
Heerstraße
100
Spree
R1
Berlin
Karl-Marx-Allee
MARZAHN-HELLERSDORF
Heerstraße
Kaiserdamm
Alt-Mahlsdorf
CHARLOTTENBURG-WILMERSDORF
51
100
100
R1
Havel
Grunewald
103
TEMPELHOF-SCHÖNEBERG
Spree
Havel
NEUKÖLLN
STEGLITZ-ZEHLENDORF
51
R1
TREPTOW-KÖPENICK

Berlin Zoom 11


WESTERN ADDITION
Alamo Square
Panhandle
HAIGHT ASHBURY
Buena Vista Park
Corona Heights Park
CASTRO
Mission Dolores Park
MISSION DISTRICT
POTRERO HILL
Sutro Open Space Reserve
Geary Blvd.
Turk Blvd.
Golden Gate Ave.
McAllister St.
Fulton St.
Hayes St.
Fell St.
Oak St.
Page St.
Haight St.
Duboce Ave.
Market St.
Valencia St.
Folsom St.
Howard St.
Central Fwy.
101
80
30
45

San Francisco Zoom 14

And links for everyone who sent in a response!

Canada, Europe, Pacific	United States
Auckland, New Zealand (https://mapzen.com/products/maps/walkabout/bike-map/#lat=-36.8764&lng=174.7715&z=13)	**Alexandria, VA (https://mapzen.com/products/maps/walkab map/#lat=38.8106&lng=-77.0617&z=14)**
Berlin, Germany (https://mapzen.com/products/maps/walkabout/bike-map/#lat=52.5162&lng=13.3922&z=13)	**Austin, TX (https://mapzen.com/products/maps/walkab map/#lat=30.2669&lng=-97.7371&z=13)**
Brampton, Canada (https://mapzen.com/products/maps/walkabout/bike-map/#lat=43.7251&lng=-79.7595&z=13)	**Berkeley, CA (https://mapzen.com/products/maps/walkab map/#lat=37.8682&lng=-122.2808&z=14)**

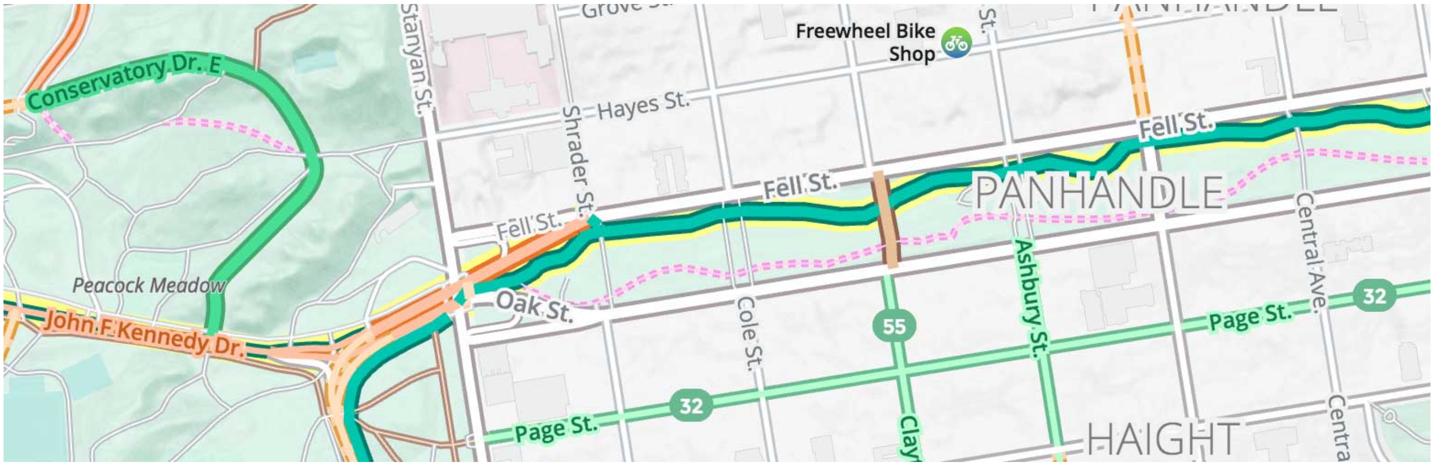
Canada, Europe, Pacific	United States
Bratislava, Slovakia (https://mapzen.com/products/maps/walkabout/bike-map/#lat=48.1492&lng=17.1181&z=14)	**Boston, MA (https://mapzen.com/products/maps/walkab map/#lat=42.3539&lng=-71.0692&z=14)**
Bremen, Germany (https://mapzen.com/products/maps/walkabout/bike-map/#lat=53.0888&lng=8.8289&z=13)	**Boulder, CO (https://mapzen.com/products/maps/walkab map/#lat=40.0198&lng=-105.2572&z=14)**
Brisbane, Australia (https://mapzen.com/products/maps/walkabout/bike-map/#lat=-27.4589&lng=153.0243&z=13)	**Brooklyn, NY (https://mapzen.com/products/maps/walkab map/#lat=40.6804&lng=-73.9667&z=14)**
Calgary, Canada (https://mapzen.com/products/maps/walkabout/bike-map/#lat=51.0562&lng=-114.0729&z=13)	**Burlington, VT (https://mapzen.com/products/maps/walkab map/#lat=44.4730&lng=-73.2062&z=14)**
Copenhagen, Denmark (https://mapzen.com/products/maps/walkabout/bike-map/#lat=55.6885&lng=12.5426&z=12)	**Chicago, IL (https://mapzen.com/products/maps/walkab map/#lat=41.8805&lng=-87.6572&z=13)**
Dublin, Ireland (https://mapzen.com/products/maps/walkabout/bike-map/#lat=53.3483&lng=-6.2725&z=13)	**Fort Collins, CO (https://mapzen.com/products/maps/walkab map/#lat=40.5561&lng=-105.0677&z=13)**
Hagen, Germany (https://mapzen.com/products/maps/walkabout/bike-map/#lat=51.3698&lng=7.4736&z=13)	**Los Angeles, CA (https://mapzen.com/products/maps/walkab map/#lat=34.0599&lng=-118.2984&z=12)**
Helsinki, Finland (https://mapzen.com/products/maps/walkabout/bike-map/#lat=60.1734&lng=24.9324&z=13)	**Minneapolis, MN (https://mapzen.com/products/maps/walkab map/#lat=44.9701&lng=-93.2717&z=13)**
Karlsruhe, Germany (https://mapzen.com/products/maps/walkabout/bike-map/#lat=49.0125&lng=8.4059&z=13)	**Mountain View, CA (https://mapzen.com/products/maps/walkab map/#lat=37.4120&lng=-122.0813&z=13)**
Lucerne, Switzerland (https://mapzen.com/products/maps/walkabout/bike-map/#lat=47.0524&lng=8.3134&z=13)	**Missoula, MT (https://mapzen.com/products/maps/walkab map/#lat=46.8726&lng=-114.0103&z=13)**
Oxford, UK (https://mapzen.com/products/maps/walkabout/bike-map/#lat=51.7531&lng=-1.2553&z=14)	**New York, NY (https://mapzen.com/products/maps/walkab map/#lat=40.7463&lng=-73.9876&z=13)**

Canada, Europe, Pacific	United States
Paris, France (https://mapzen.com/products/maps/walkabout/bike-map/#lat=48.8512&lng=2.3348&z=13)	Oakland, CA (https://mapzen.com/products/maps/walkab map/#lat=37.8012&lng=-122.2564&z=13)
Recife, Brazil (https://mapzen.com/products/maps/walkabout/bike-map/#lat=-8.0581&lng=-34.9077&z=13)	Portland, OR (https://mapzen.com/products/maps/walkab map/#lat=45.5330&lng=-122.6526&z=13)
Reykjavik, Iceland (https://mapzen.com/products/maps/walkabout/bike-map/#lat=64.1311&lng=-21.8870&z=13)	San Francisco, CA (https://mapzen.com/products/maps/walkab map/#lat=37.7639&lng=-122.4435&z=13)
Saskatoon, Canada (https://mapzen.com/products/maps/walkabout/bike-map/#lat=52.1268&lng=-106.6681&z=13)	Santa Barbara, CA (https://mapzen.com/products/maps/walkab map/#lat=34.4294&lng=-119.7335&z=13)
Thessaloniki, Greece (https://mapzen.com/products/maps/walkabout/bike-map/#lat=40.6436&lng=22.9309&z=13)	Santa Clara county, CA (https://mapzen.com/products/maps/walkab map/#lat=37.3400&lng=-121.9383&z=12)
Toulouse, France (https://mapzen.com/products/maps/walkabout/bike-map/#lat=43.6044&lng=1.4419&z=13)	Saratoga Springs, UT (https://mapzen.com/products/maps/walkab map/#lat=40.3538&lng=-111.9119&z=13)
Turin, Italy (https://mapzen.com/products/maps/walkabout/bike-map/#lat=45.0705&lng=7.6868&z=13)	Seattle, WA (https://mapzen.com/products/maps/walkab map/#lat=47.6053&lng=-122.3372&z=13)
Umeå, Sweden (https://mapzen.com/products/maps/walkabout/bike-map/#lat=63.8222&lng=20.2773&z=13)	Walnut Creek, CA (https://mapzen.com/products/maps/walkab map/#lat=37.9019&lng=-122.0645&z=13)
Windsor, Ontario (https://mapzen.com/products/maps/walkabout/bike-map/#lat=42.2987&lng=-83.0154&z=13)	
Xixón, Spain (https://mapzen.com/products/maps/walkabout/bike-map/#lat=43.5335&lng=-5.6636&z=14)	

New Colors!

We got *lots* of feedback about color, and as a result we've simplified the scheme to **orange and green**. By breaking the two colors into various shades, **darker to lighter**, we are able to show even more details, with darker being "better" in the hierarchy of safety.

For example, see San Francisco below: dark green is used for cycleways (protected bike paths) in the Panhandle, while medium green shows "greenways" in Golden Gate Park (more on those in a minute), and light green indicates rides on lower-traffic minor roads.

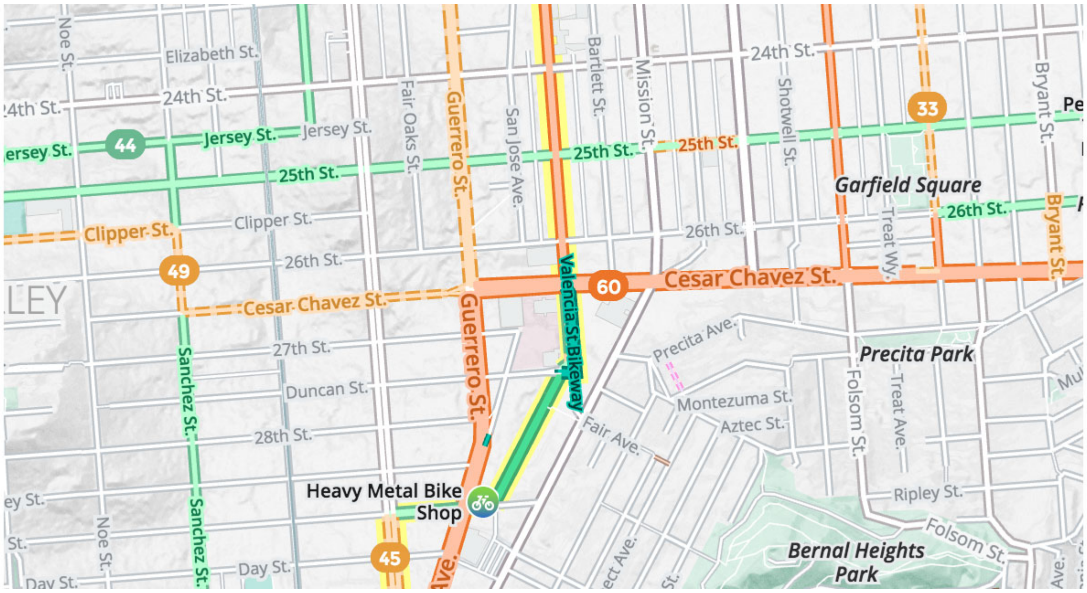

Conservatory Dr. E
Stanyan St.
Grove St.
Freewheel Bike Shop
St.
Hayes St.
Shrader St.
Fell St.
Fell St.
Fell St.
PANHANDLE
Peacock Meadow
John F. Kennedy Dr.
Oak St.
Cole St.
55
Ashbury St.
Central Ave.
32
Page St.
32
Page St.
HAIGHT

Three shades of green in the hierarchy: dark green cycleways in the Panhandle, medium green for the greenway on Conservatory Dr. in Golden Gate Park, and light green for Page St.

High Traffic Roads

The biggest change when considering colors on this new iteration of the map is the **kind of road (https://wiki.openstreetmap.org/wiki/Key:highway)** in addition to the **bicycle (https://wiki.openstreetmap.org/wiki/Bicycle)** data.

In the first version of the bike map, colors showed bike lanes and routes *regardless* of road type. But on this new version, we've addressed one major point of feedback, which is changing the way we emphasize high-traffic arterial roads versus more bike-friendly residential streets. **Orange** is now used for arterial roads, which generally have more car traffic. We're also using the **dashing element** as a warning if there are no bike lanes on those high-traffic roads.


Elizabeth St.
24th St.
Jersey St.
25th St.
Clipper St.
26th St.
Cesar Chavez St.
27th St.
Duncan St.
28th St.
Sanchez St.
Noe St.
Fair Oaks St.
Guerrero St.
San Jose Ave.
Bartlett St.
Mission St.
Shotwell St.
Bryant St.
Valencia St.Bikeway
Garfield Square
Treat Wy.
Precita Ave.
Precita Park
Montezuma St.
Aztec St.
Fair Ave.
Folsom St.
Treat Ave.
Ripley St.
Heavy Metal Bike Shop
Day St.
Bernal Heights Park
44
49
60
33
45

Cesar Chavez St. between Guerrero and Bryant is a major arterial with bike lanes shown in solid orange casing, but east of Guerrero bicyclists "share" the road with cars on a sign-only route shown with a lighter, dashed orange casing.

Brown remains the color used for tracks. Unpaved roads have now been added to this track category.

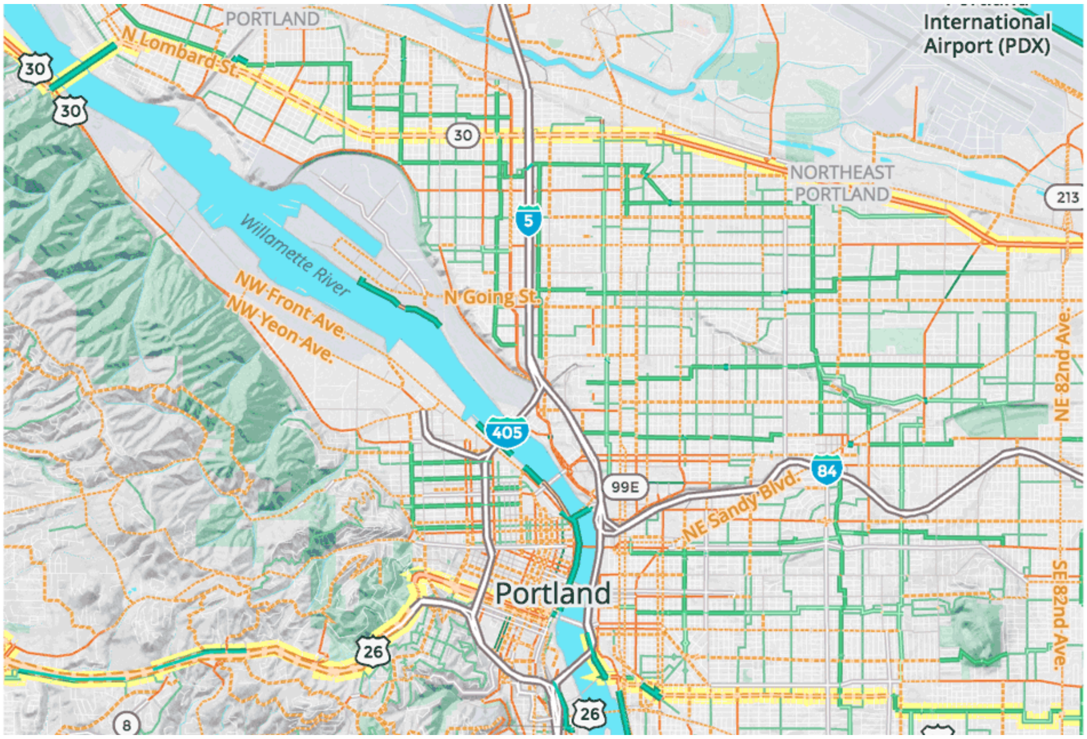
We've also removed the **yellow highlight** from local networks. Instead, we've emphasized the local network by adjusting line widths. This reduces the confusion of highlights and tier colors mixing visually. (Note we still highlight regional, national, and international networks.)

Greenway/Living Streets

After careful study and comparisons of the dozens of city bike maps you sent us, we've paid particular attention to "living street" and neighbourhood greenways, emphasizing their width and giving them the medium green shade, as well as revealing them at earlier zooms.

If a road is tagged as residential, unclassified or living street, as well as being part of a local cycle network, it will be shown in our new greenway category. Below is a more detailed definition of greenway or "bicycle boulevard" from the book ***Urban Bikeway Design Guide by the National Association of City Transportation Officials*** **(https://nacto.org/publication/urban-bikeway-design-guide/)**.

> *“Bicycle boulevards are streets with low motorized traffic volumes and speeds, designated and designed to give bicycle travel priority. Bicycle Boulevards use signs, pavement markings, and speed and volume management measures to discourage through trips by motorvehicles and create safe, convenient bicycle crossings of busy arterial streets.”*

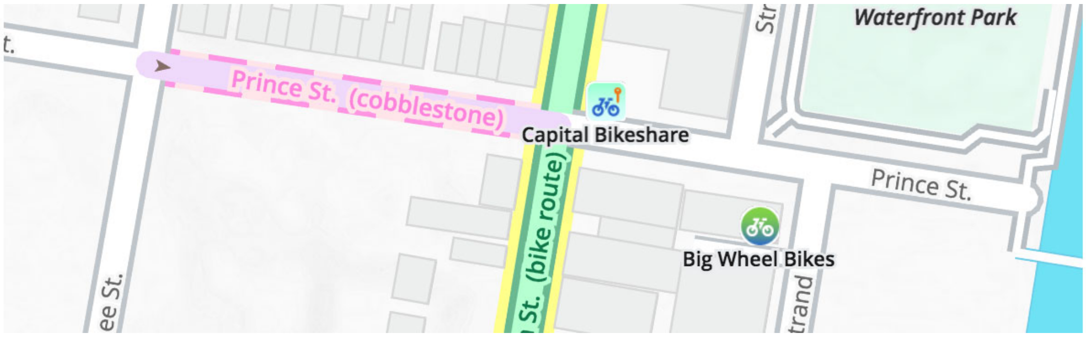

PORTLAND
N Lombard St.
30
30
30
International Airport (PDX)
NORTHEAST PORTLAND
213
5
Willamette River
NW Front Ave.
NW Yeon Ave.
N Going St.
NE 82nd Ave.
405
84
99E
NE Sandy Blvd.
Portland
26
26
SE 82nd Ave.
8

Zooming in on greenways in Portland: NE Tillamook Street, NE Morris Street, and NE Siskiyou Street

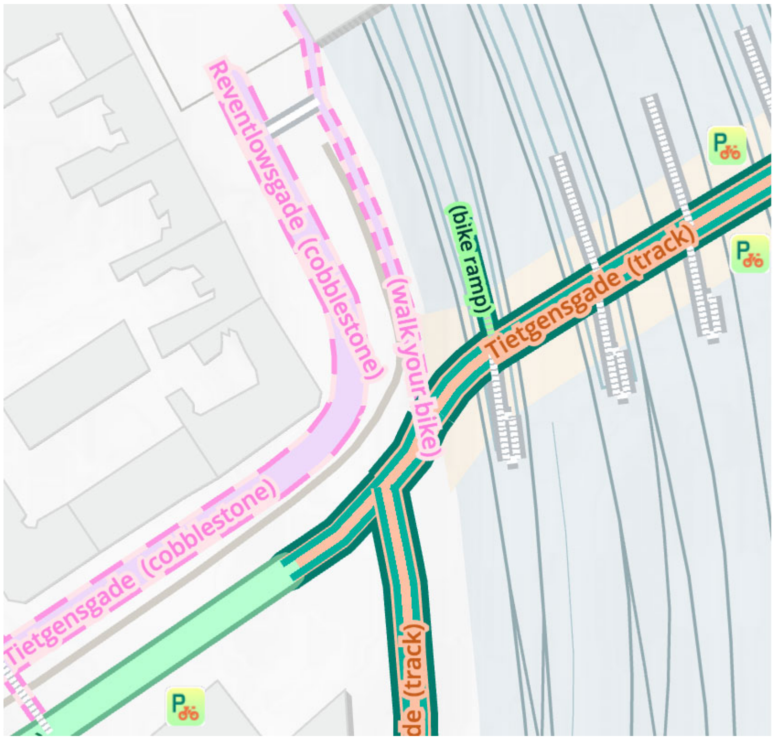
Learn more on how greenways are being used in Portland (https://www.portlandoregon.gov/transportation/article/348902).

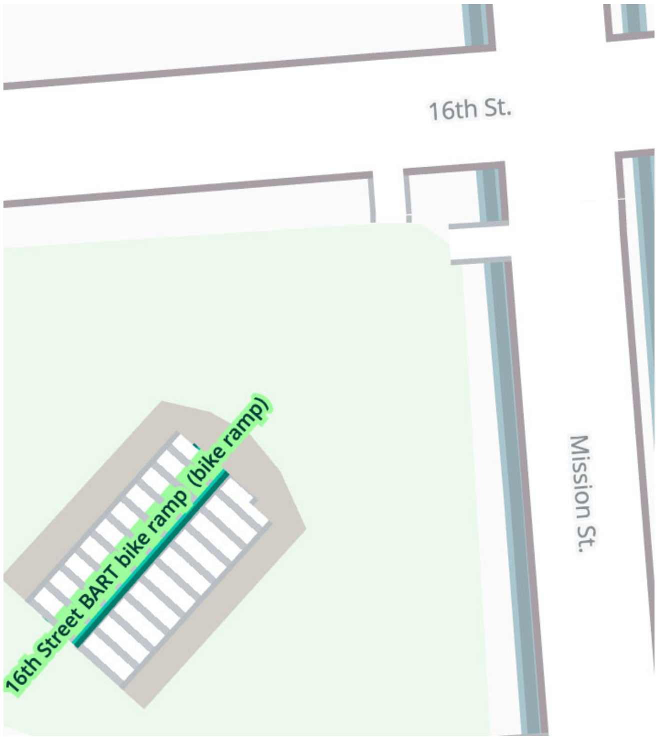
Other Road Color Changes

Cobblestone streets have been added to the bike map. (These are not fun.) See Prince Street in Alexandria below.


t.
Prince St. (cobblestone)
Str
Waterfront Park
Capital Bikeshare
Prince St.
St. (bike route)
Big Wheel Bikes
ee St.
trand

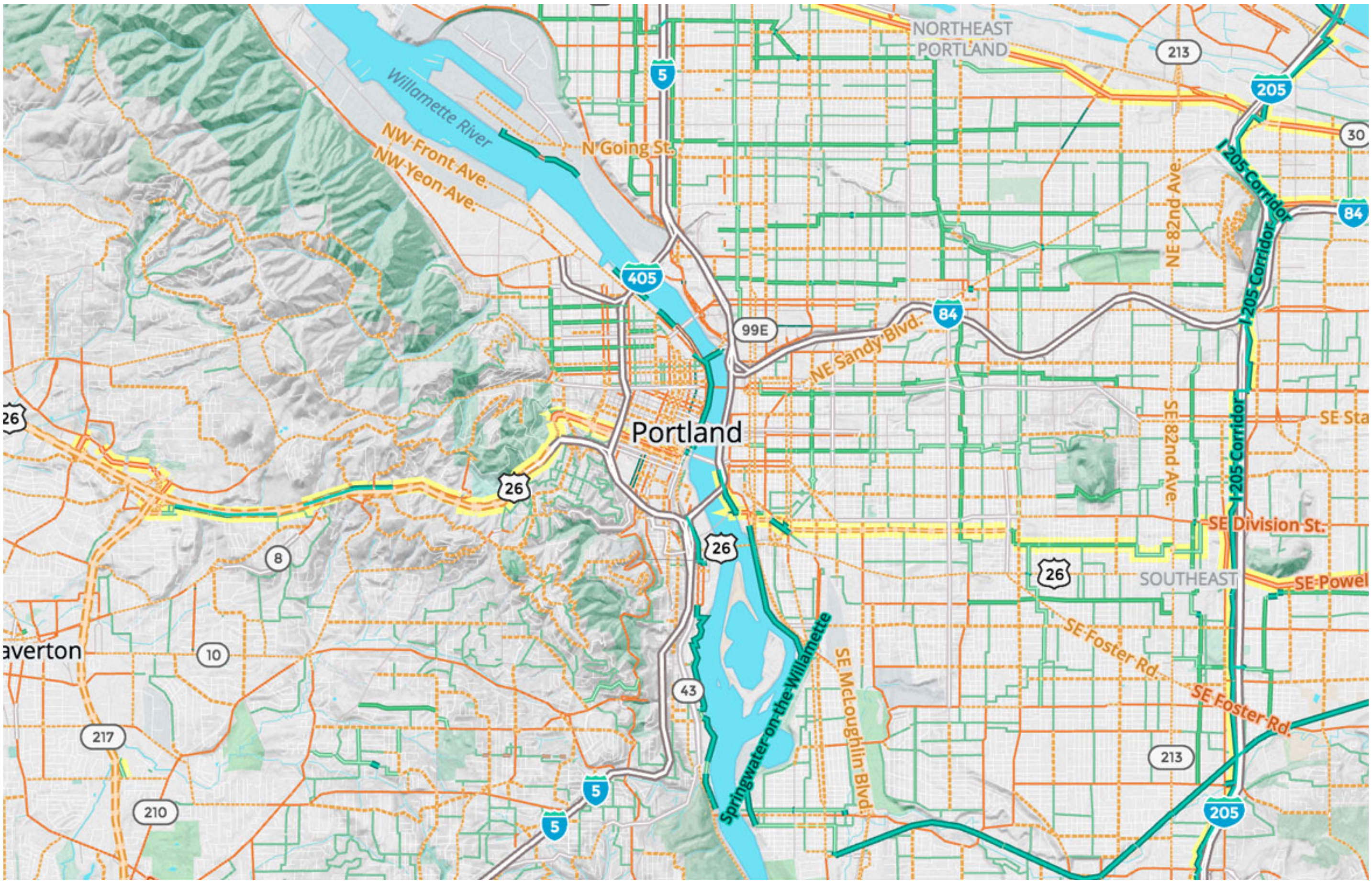
Bike Ramps have been added as well. (These are fun!) See Copenhagen below, along with San Francisco.

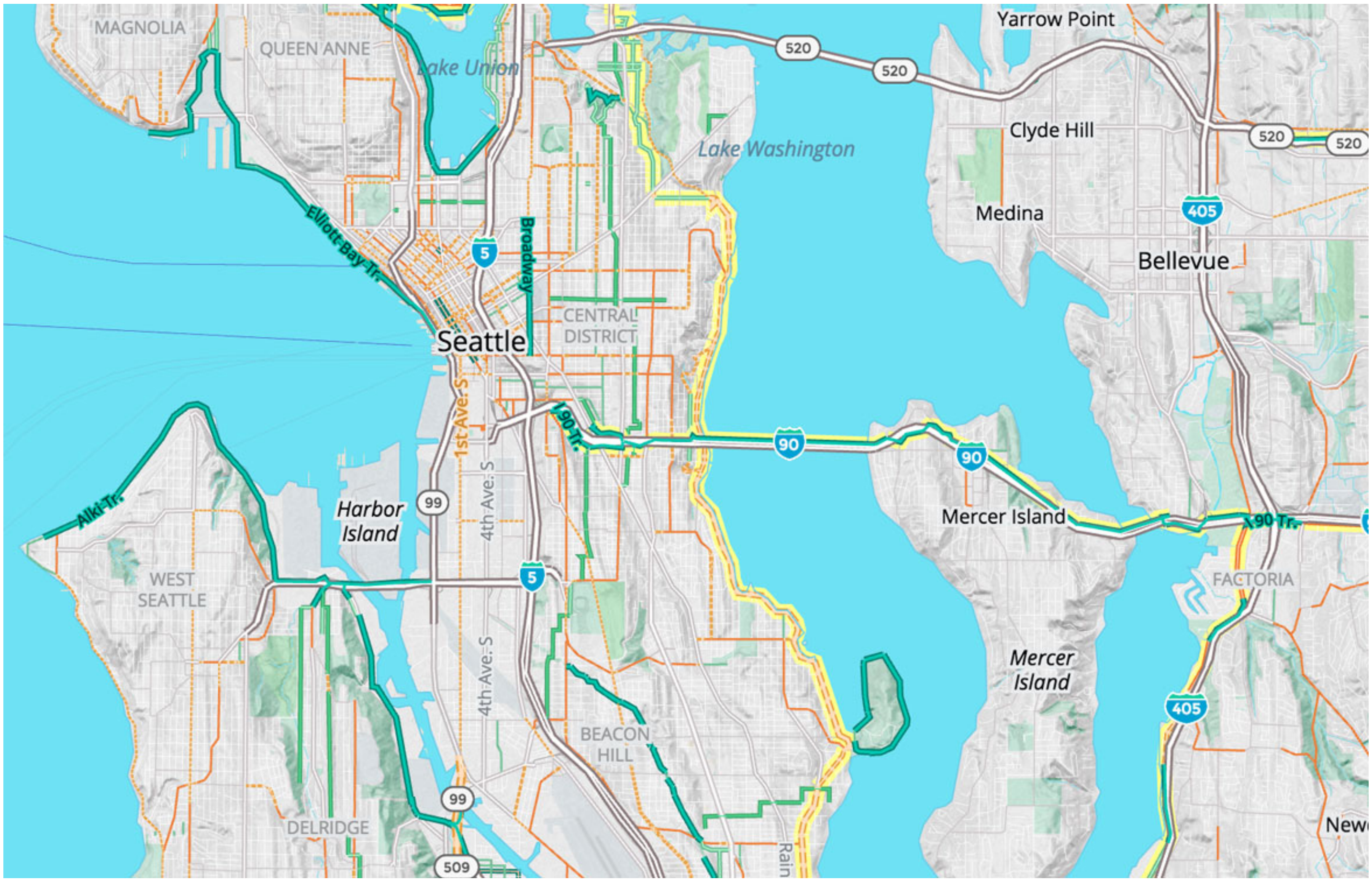

Reventlowsgade (cobblestone)
(walk your bike)
(bike ramp)
Tietgensgade (track)
Tietgensgade (cobblestone)
de (track)


16th St.
Mission St.
16th Street BART bike ramp (bike ramp)

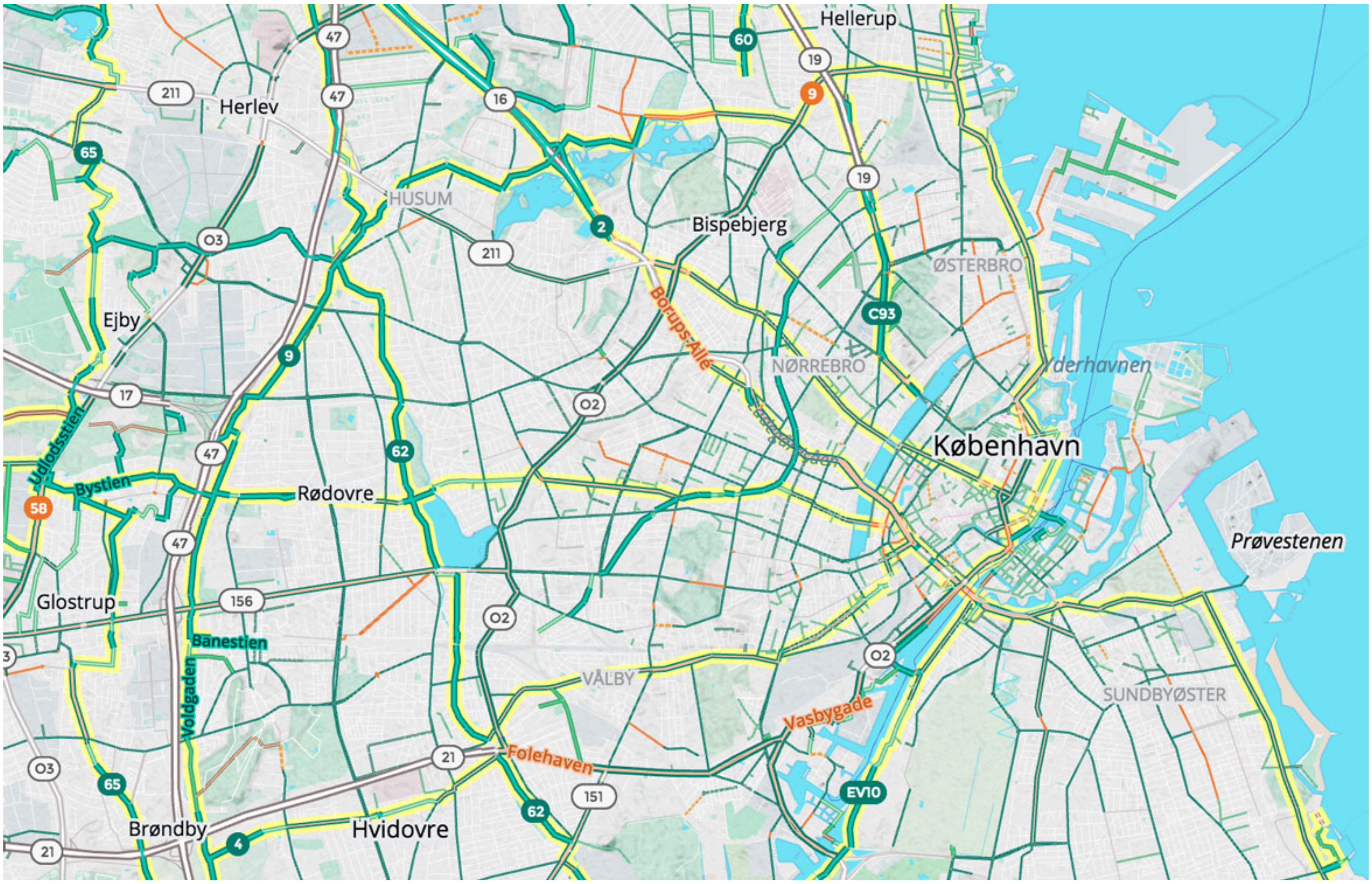
Bike Networks and Improved Coverage at Mid- to Low-Zooms

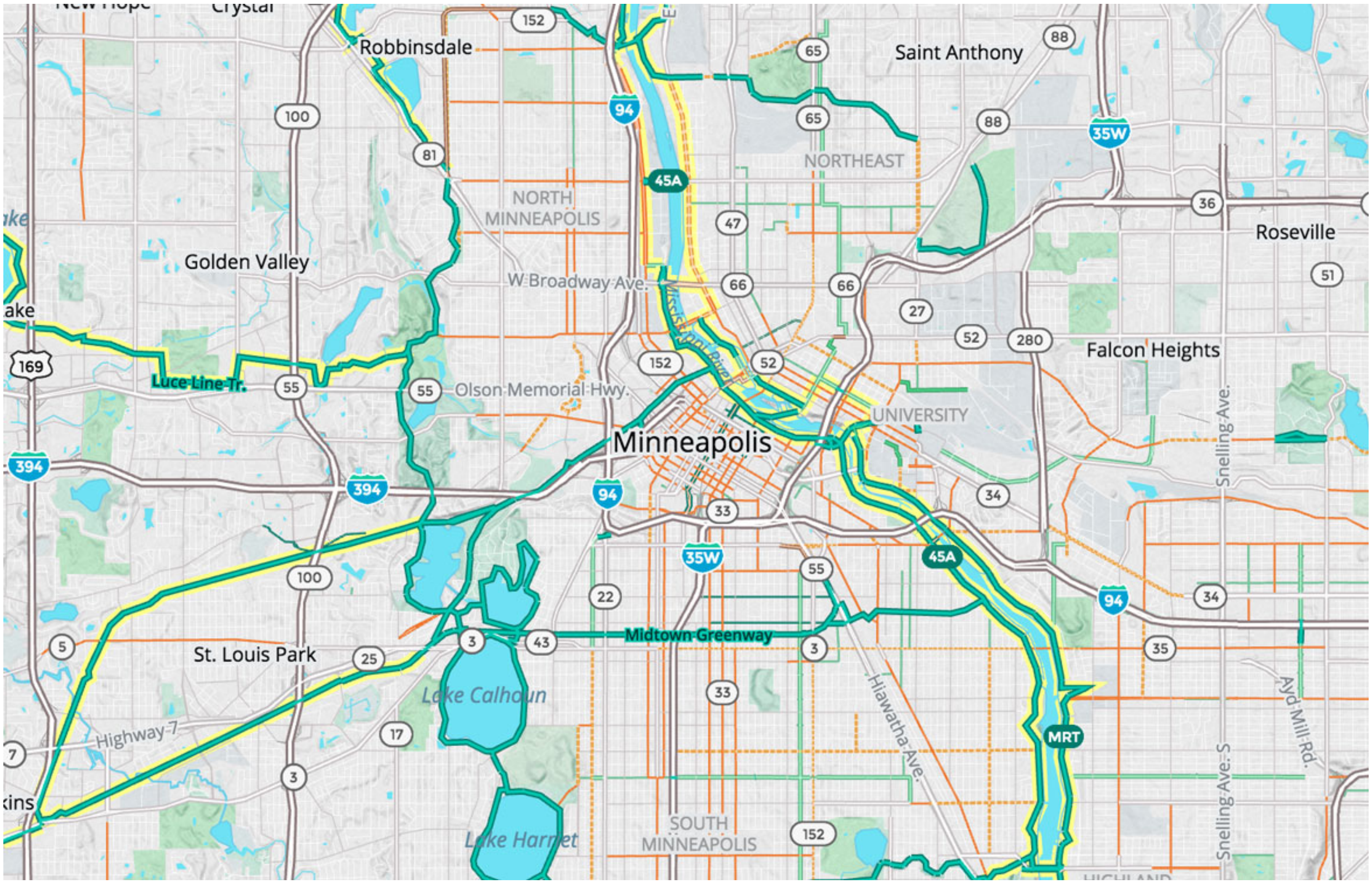
We found out from those who took the survey that almost everyone bikes in the city (97%). Half also bike outside the city, while 15% like to mountain bike, and 17% enjoy bike touring (through the country and along the coast). Given this feedback, we've focused a lot of our work on city overviews and made improvements connecting network views at the mid- to low-zooms. Seattle and Portland are good examples of how local networks and greenways are more readily visible.

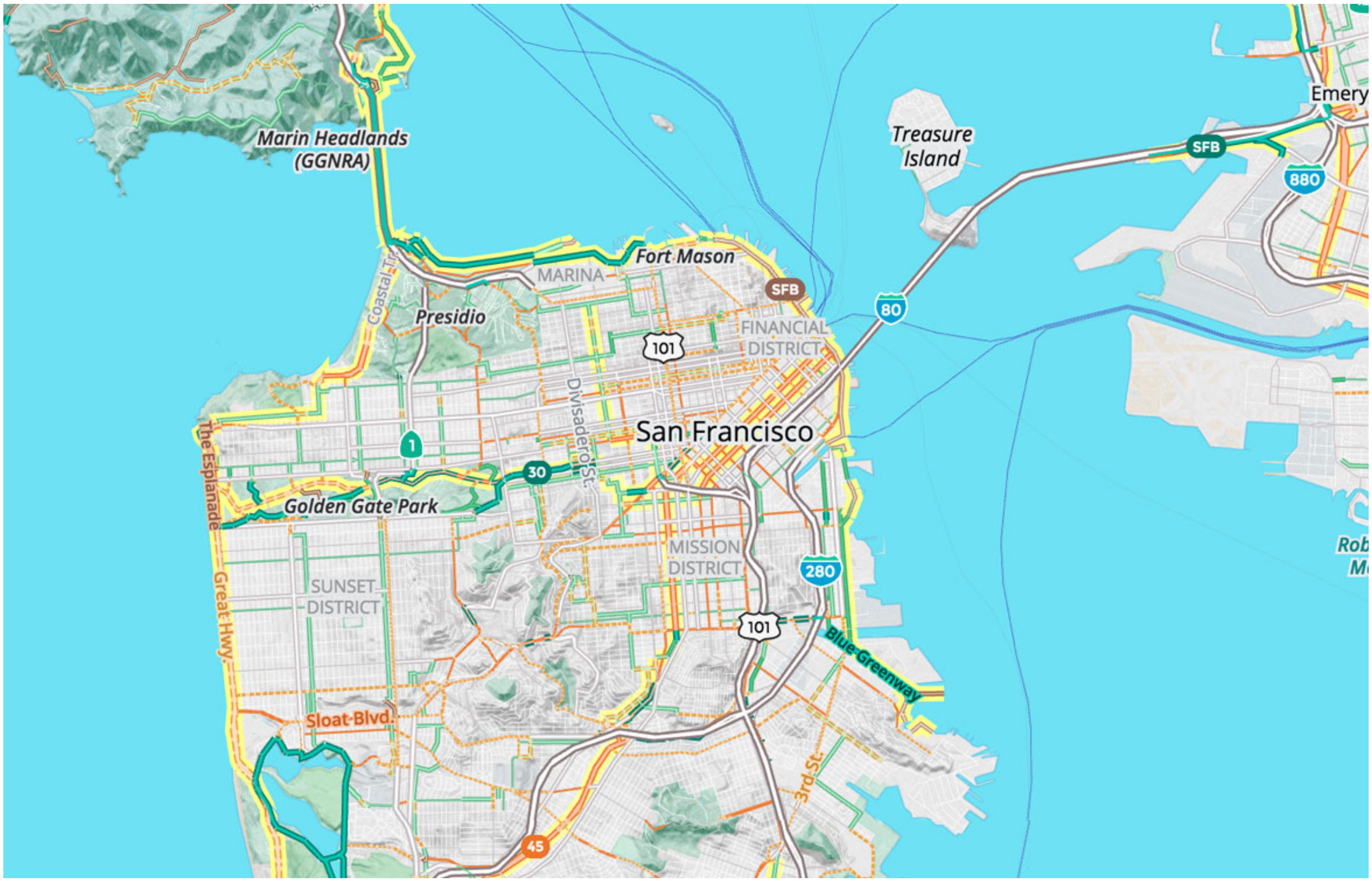

NORTHEAST
PORTLAND
213
205
30
84
I-205 Corridor
NE 82nd Ave.
N Going St.
Willamette River
NW Front Ave.
NW Yeon Ave.
5
405
99E
84
NE Sandy Blvd.
Portland
26
SE 82nd Ave.
SE Sta
I-205 Corridor
SE Division St.
SOUTHEAST
SE Powel
26
26
8
26
averton
10
43
Springwater on the Willamette
SE McLoughlin Blvd.
SE Foster Rd.
SE Foster Rd.
217
213
210
5
5
205


MAGNOLIA
QUEEN ANNE
Lake Union
520
520
Yarrow Point
Clyde Hill
520
520
Lake Washington
Medina
405
Bellevue
Elliott Bay Tr.
5
Broadway
CENTRAL
DISTRICT
Seattle
1st Ave. S
I-90 Tr.
90
90
Mercer Island
I-90 Tr.
Alki Tr.
Harbor
Island
99
4th Ave. S
WEST
SEATTLE
FACTORIA
5
4th Ave. S
Mercer
Island
405
BEACON
HILL
99
DELRIDGE
509
Rain
New

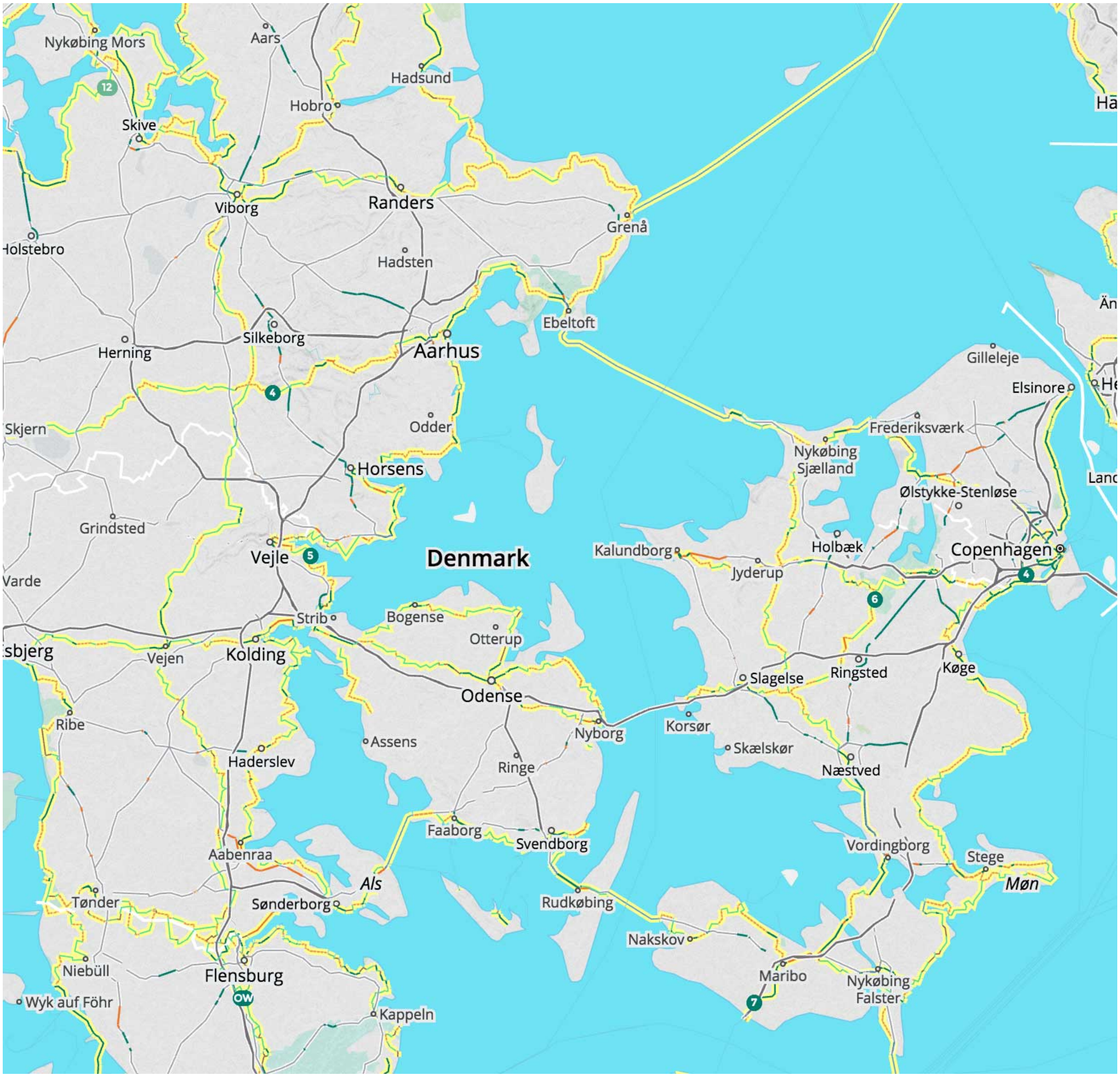
Below are more zoom 12 views of Copenhagen, Minneapolis, and San Francisco. Notice how bike infrastructure vary from city to city, from the overall green of Copenhagen's numerous cycle tracks, to the green rings around Minneapolis and the Midtown Greenway cutting across the city, along with San Francisco's primarily orange higher-traffic downtown area.


Hellerup
Herlev
HUSUM
Bispebjerg
ØSTERBRO
Ejby
Borups Allé
NØRREBRO
Nordhavnen
København
Udlodsstien
Bystien
Rødovre
Prøvestenen
Glostrup
Banestien
VALBY
Voldgaden
SUNDBYØSTER
Vasbygade
Folehaven
Brøndby
Hvidovre
47
60
19
211
16
9
65
2
O3
C93
17
O2
62
58
156
21
151
EV10
4


Robbinsdale
Saint Anthony
NORTHEAST
NORTH MINNEAPOLIS
Golden Valley
Roseville
W Broadway Ave.
Falcon Heights
Luce Line Tr.
Olson Memorial Hwy.
UNIVERSITY
Minneapolis
Snelling Ave.
Midtown Greenway
St. Louis Park
Lake Calhoun
Highway 7
Hiawatha Ave.
Ayd Mill Rd.
Lake Harriet
SOUTH MINNEAPOLIS
MRT
45A
94
35W
394
169
100
55
152
65
88
81
47
66
36
51
27
52
280
34
33
22
43
3
5
25
17
7
Snelling Ave. S


Marin Headlands (GGNRA)
Treasure Island
Emery
SFB
880
Fort Mason
MARINA
Coastal Tr.
Presidio
FINANCIAL DISTRICT
80
101
Divisadero St.
San Francisco
The Esplanade
1
30
Golden Gate Park
MISSION DISTRICT
280
SUNSET DISTRICT
Great Hwy.
Blue Greenway
Sloat Blvd.
3rd St.
45

Zooming further out, you can see national and international networks thanks to **significant enhancements (https://github.com/tilezen/vector-datasource/releases/tag/v1.3.0)** in Mapzen's vector tiles. And in Denmark we can see bike routes on ferries!

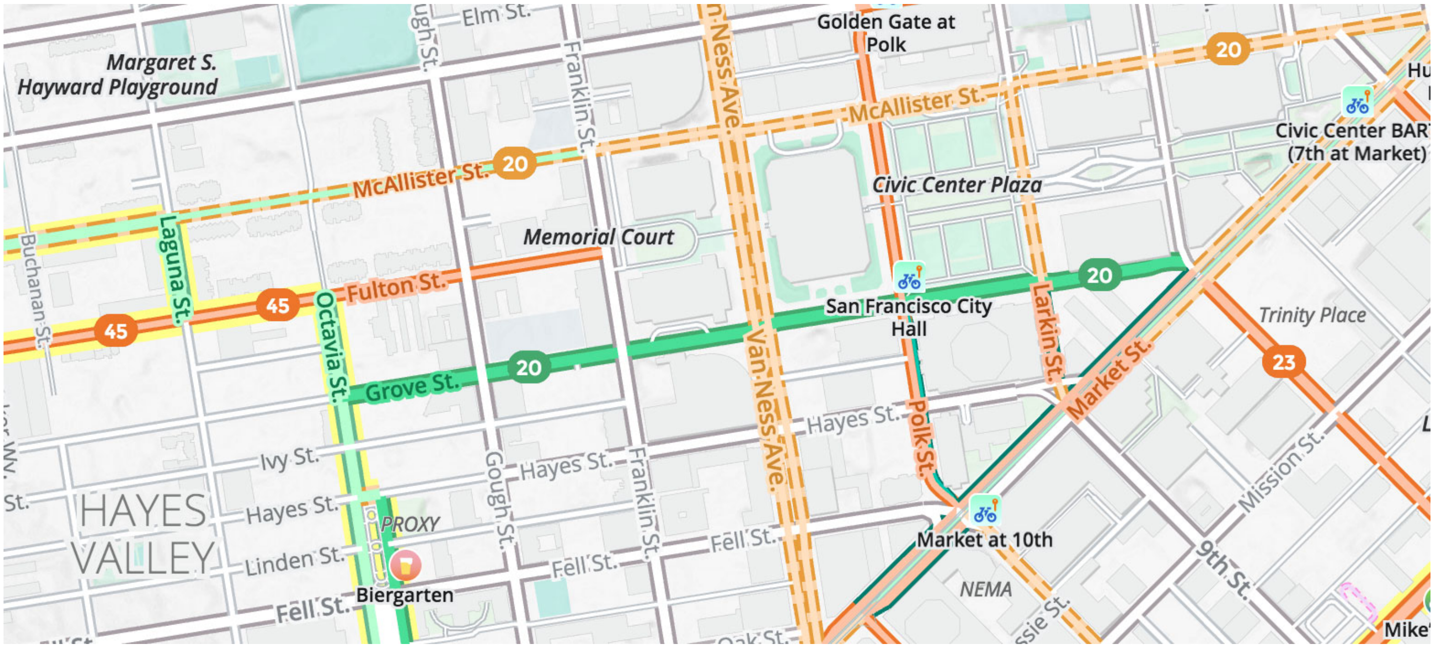

Nykøbing Mors
Aars
Hadsund
12
Hobro
Skive
Viborg
Randers
Grenå
Holstebro
Hadsten
Ebeltoft
Herning
Silkeborg
Aarhus
Gilleleje
4
Elsinore
Skjern
Odder
Frederiksværk
Nykøbing Sjælland
Horsens
Ølstykke-Stenløse
Grindsted
Holbæk
Kalundborg
Copenhagen
Vejle
5
Denmark
Jyderup
4
Varde
6
Strib
Bogense
Otterup
Esbjerg
Vejen
Kolding
Slagelse
Ringsted
Køge
Odense
Ribe
Nyborg
Korsør
Assens
Skælskør
Ringe
Næstved
Haderslev
Faaborg
Svendborg
Vordingborg
Aabenraa
Stege
Als
Møn
Tønder
Sønderborg
Rudkøbing
Nakskov
Niebüll
Flensburg
Maribo
Nykøbing Falster
OW
7
Wyk auf Föhr
Kappeln

Bicycle Data From City to City

Regardless of the quality of the infrastructure, many cities are looking great (San Francisco; Portland; Washington, D.C.; Copenhagen). Some are a work in progress. Some are missing entirely, like Tokyo, but you can change that in OpenStreetMap! Check out our **Targeted Editing post on cycleways (https://mapzen.com/blog/targeted-editing-cycleways/)** to learn more, and get your bike network on the map.

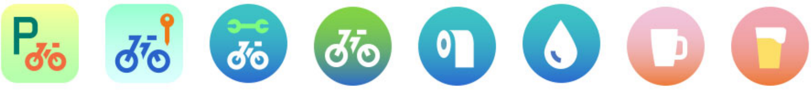
Bike Shields

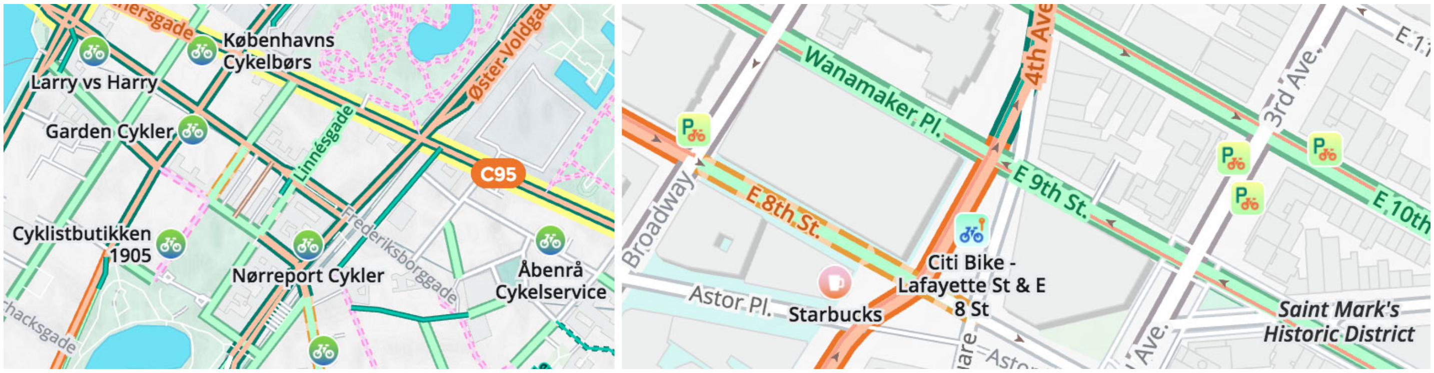
Some cities have numbered bike routes, so we’ve added bike shields using new data in Mapzen vector tiles!


Elm St.
Golden Gate at Polk
Margaret S. Hayward Playground
McAllister St.
20
Civic Center BART (7th at Market)
Civic Center Plaza
Memorial Court
Buchanan St.
Laguna St.
Fulton St.
45
Octavia St.
Grove St.
San Francisco City Hall
Larkin St.
Trinity Place
23
Market St.
Van Ness Ave.
Hayes St.
Polk St.
Mission St.
Ivy St.
Gough St.
Franklin St.
HAYES VALLEY
Linden St.
PROXY
Fell St.
Biergarten
Market at 10th
NEMA
9th St.

Bike Icons

Bike icons have new colors and are now enclosed in attractive circular badges.




København Cykelbørs
Larry vs Harry
Garden Cykler
Øster Voldgade
Linnésgade
C95
Cyklistbutikken 1905
Nørreport Cykler
Frederiksborggade
Åbenrå Cykelservice
Wanamaker Pl.
4th Ave.
3rd Ave.
E 9th St.
E 8th St.
E 10th
Broadway
Citi Bike - Lafayette St & E 8 St
Astor Pl.
Starbucks
Saint Mark's Historic District

Bike icons in Copenhagen and New York

bicycle=designated

The future of bike tags is already here, it's just not very evenly distributed in OpenStreetMap. We now show include `bicycle=designated` in our styling, which is commonly used in German cities, Portland, and many other places (especially for bike routes).

Get the Bike Overlay

Just like **enabling languages (https://mapzen.com/blog/languages-of-india/)**, it's very easy to turn on the bike overlay after importing the Walkabout basemap style in Tangram – just import the basemap and add a few lines! The Cartography **docs (https://mapzen.com/documentation/cartography/api-reference/#bike-overlay)** include the following example and bonus map legend.

Set the required **Mapzen API key (https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys)** to access vector and terrain tiles, disable the path overlay (which is on by default), and enable the optional bike overlay:

```
import: https://mapzen.com/carto/walkabout-style/5/walkabout-style.zip

global:
  sdk_mapzen_api_key: mapzen-xxxxxx          # your API key here
  sdk_path_overlay: false
  sdk_bike_overlay: true
```

If you're using **mapzen.js (https://github.com/mapzen/mapzen.js)**:

```
var map = L.Mapzen.map('map', {
  center: [40.8041, -124.1506],
  zoom: 15,
  maxZoom: 20,
  tangramOptions: {
    scene: {
      import: L.Mapzen.BasemapStyles.Walkabout,
      global: {
        sdk_mapzen_api_key: 'mapzen-xxxxxx',  // your API key here
        sdk_path_overlay: false,
        sdk_bike_overlay: true
      }
    }
  }
});
```

Work In Progress

We're still working on these other suggestions we've received from you:

- Color blind mode
- Construction blockages
- Display steepness of grade
- Routing widget

The Future

We hope you like the color and styling improvements! We love getting your feedback. Let's keep the conversation going as we continue to make improvements.

Let us know (mailto:hello@mapzen.com&subject=%22bike%20map%22) what you think. And **our poll is still open (https://docs.google.com/forms/d/1pAi1hWj8tW7ssxo9lOjW1xtcRYo3RUswvVQSmTxbkqg)**!

Safe biking!

· 11 May 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

Mapping the ages of buildings in Seoul

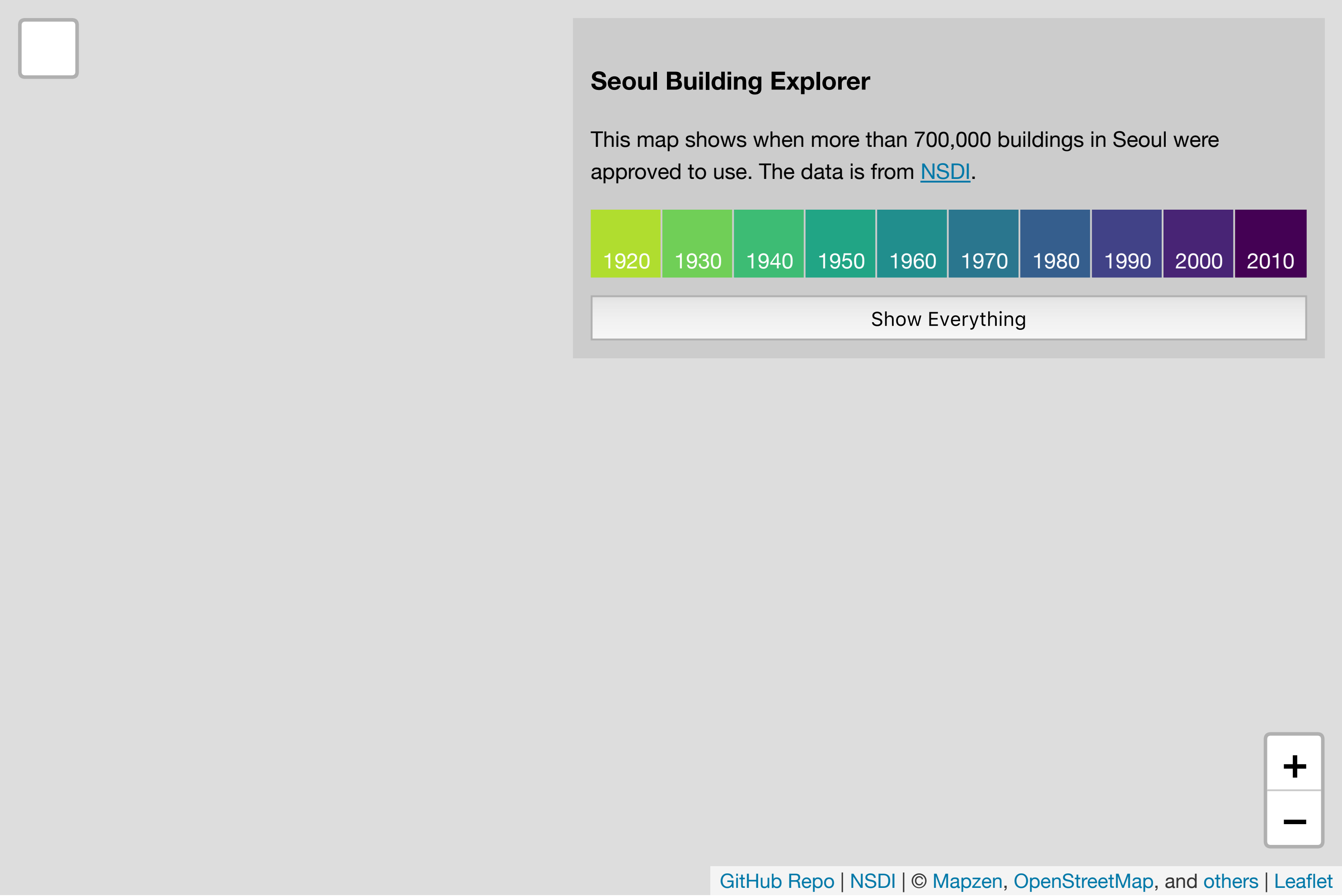
data (/tag/data) tangram (/tag/tangram) demo (/tag/demo)

Mapping of the ages of buildings in Seoul

South Korea has been releasing lots of great open data through its **National Spatial Data Infrastructure Portal (NSDI) (http://nsdi.go.kr/)**. It includes more than 750,000 buildings in Seoul, including their shape, height, neighborhood and approval date. Inspired by **built:LA (http://cityhubla.github.io/LA_Building_Age/#12/34.0267/-118.2621)**, I decided to make a dynamic map showing the age of buildings in Seoul with Tangram.

When the map is zoomed in, you can see the shape and height of individual buildings, colored by the year of their construction. You can also click on individual building to find out more details. As you zoom out, the map changes to dots, and then shows age data aggregated by neighborhood. Click on a color block in the legend to just show buildings from a specific decade!

Can you guess which part of Seoul is the old downtown area by looking at this map?



Seoul Building Explorer

This map shows when more than 700,000 buildings in Seoul were approved to use. The data is from NSDI.

1920
1930
1940
1950
1960
1970
1980
1990
2000
2010
Show Everything
+
−
GitHub Repo | NSDI | © Mapzen, OpenStreetMap, and others | Leaflet

Full Screen Map (https://hanbyul-here.github.io/seoul-building-explorer/#lat=37.56538&lng=126.98021&z=16.0250) || ***한국어로 지도 보기 (https://hanbyul-here.github.io/seoul-building-explorer/#lat=37.56538&lng=126.98021&z=16.0250&lang=kr)***

FF development log

Post-process of data and custom tiles

The original data from NSDI was in shapefile format. In QGIS, I converted it into a huge GeoJSON file. The projection of the shapefile was slightly off and encoding of the data was `euc-kr`, so I re-encoded to `utf-8`, and reprojected it when I exported it to GeoJSON.

The exported GeoJSON was over 300mb — too big to be loaded at once — so it had to be tiled. All of these datasets were built into tiles with **TileStache (http://tilestache.org/)**, hosted in S3 buckets. To make loading time for each tile quicker, we split it into 3 datasets based on building age.

In case you need inspiration, **here is how I configured TileStache (https://gist.github.com/hanbyul-here/f856d8d466604e0fcb40a6827658072e)** to generate the static vector tile set. If your file is not as big as the one used here, but still big enough to be tiled, **geojson2mvt (https://github.com/NYCPlanning/geojson2mvt)** looks like a good option.

Looking at the shapes of buildings is interesting, but only above zoom level 15. For zoom 13-14, John Oram and I made centroids of each building. QGIS was surprisingly slow, but thanks to **turf.js (https://github.com/turf-junkyard/turf-centroid)** we were able to **quickly generate them (https://gist.github.com/hanbyul-here/38e964581e7a84efd0da0bf1b7a12f2f)** and strip out extra fields we didn't need. (Watch out for null geometries!)

As I played around with the slippy map, it made me want to see how old buildings were by neighborhood. Yet again, NSDI delivers! I grabbed their neighborhood data and aggregated the building age within each *"dong"*, or neighborhood. There are many ways to do this — I approached the problem using a simple **node script (https://gist.github.com/hanbyul-here/7dc7d16a910f264c09bfcfe34a8ef7a2)**.

Colors and interactions

I used the **Virdis color palette (https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html)** to have linear, but distinctive colors for each decade. (I found out about this through a kind person on Twitter. Who said Twitter is a waste of life?) These colors could have be added as a global variable in yaml, but they were saved as a hex value array in a separate script so that they could also be used on the javascript side. You can include the script you need with `scripts` attribute in Tangram.

```
sources:
  seoul-buildings-2345:
    type: GeoJSON
    url: https:/tile/address/{z}/{x}/{y}.geojson
    scripts: ['./virdisColors.js']
```

In `virdisColors.js`, we define the global variable to which the array of colors is assigned.

```
  /* virdisColors.js */
  var viridis = ["#440154", "#482475", "#414287", "#355e8d", "#2a768e", "#218e8d", "#21a5
```

We can also access `virdis` outside Tangram by including `viridis.js` :

```
<script src='virdis.js'></script>
<script>
  for (var i = viridis.length-1; i > -1; i--) {
    var colorBlock = L.DomUtil.create('div');
    colorBlock.style.backgroundColor = viridis[i];
  }
</script>
```

Finally, the index number of the array was used to color each decade:

```
global:
  matching:
      filter: global.filter
      draw:
        polygons:
          color: |
            function () {
              var arr = feature['year'].slice(0, 3);
              var age = 201 - parseInt(arr);
              return viridis[age];
            }
layers:
  buildings-2345:
    data: {source: seoul-buildings-2345}
    matching: global.matching
    draw:
      polygons:
        interactive: true
        order: 3001
        /* This color will be used when the feature is not selected */
        color: #333
```

The interactions between UI components and the map were built using Tangram’s global variable and filtering functions. For example, to highlight a specific decade, the script sets a global variable in Tangram when the color block is clicked on.

```
colorBlock.addEventListener('click', function () {
  scene.config.global.age = this.getAttribute('year');
  /* Tangram config needs to be updated */
  scene.updateConfig();
})
```

Then the filter block of yaml (which always returns `true` when no age variable is set) starts returning only the features that match the selected decades.

```
filter: |
    function () {
      if (!global.age) {
        return true;
      } else {
        /* The date of approval is saved in yyyymmdd string format */
        var featureYear = feature['year'].slice(0, 3);
        return (featureYear == global.age);
      }
    }
```

Check out the YAML here! (https://github.com/hanbyul-here/seoul-building-explorer/blob/gh-pages/assets/date.yaml)

I made sure all the text we needed was in one JSON file so it could be easily swapped into other languages. You can check out this and all the front-end parts of the map in **its repo (https://github.com/hanbyul-here/seoul-building-explorer)**.

Find building data for your city and use Tangram to make your own map!

· 15 May 2017 ·

Hanbyul Jo

Hanbyul does front-end that makes maps and candies.

© 2017 Mapzen

Statistics and validation reports for GTFS feeds in Transitland

transitland (/tag/transitland) mobility (/tag/mobility)

The development of GTFS (General Transit Feed Specification) over the past 10 years has ushered in a “Golden Age” of open transit data, with adoption by thousands of transit operators across the world. However, as GTFS has spread, a number of questions have arisen about best data practices, how to resolve ambiguities in describing complex transit systems, and how to use GTFS to solve increasingly challenging questions in urban planning and transportation.

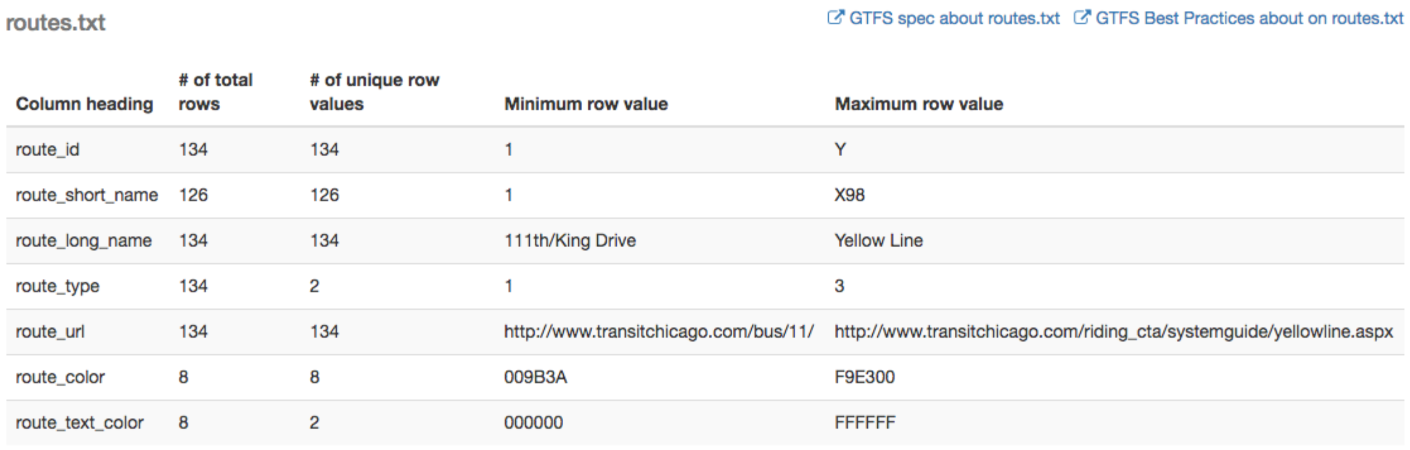
Mapzen has been participating in the Rocky Mountain Institute’s **collaborative effort to improve GTFS creation practices (http://gtfs.org)**. To help, we’ve been adding functionality to **Transitland (https://transit.land)** to collect information on feeds at scale. Transitland currently includes over 9,600 archived GTFS files from more than 800 sources, and the platform now automatically computes four different types of reports on each of these feed versions:

- Feed statistics
- Service-level histograms
- Google FeedValidator output
- Conveyal gtfs-lib output

These reports are available through the **Transitland Dispatcher (https://transit.land/dispatcher/feeds)** management interface as well as through the **Transitland Datastore API (https://transit.land/documentation/datastore/feeds.html)**.

Feed statistics

Whenever Transitland fetches a feed, it now summarizes the files present in each archive, the columns in each file, and the values in each column:



routes.txt

GTFS spec about routes.txt GTFS Best Practices about on routes.txt

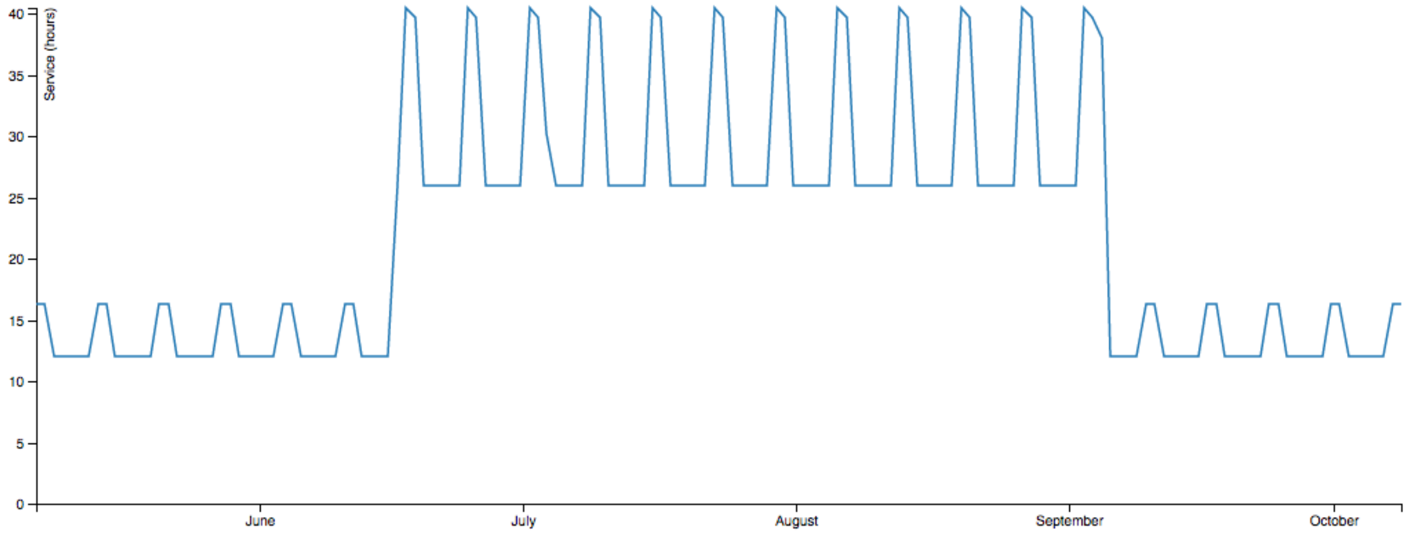
Column heading	# of total rows	# of unique row values	Minimum row value	Maximum row value
route_id	134	134	1	Y
route_short_name	126	126	1	X98
route_long_name	134	134	111th/King Drive	Yellow Line
route_type	134	2	1	3
route_url	134	134	http://www.transitchicago.com/bus/11/	http://www.transitchicago.com/riding_cta/systemguide/yellowline.aspx
route_color	8	8	009B3A	F9E300
route_text_color	8	2	000000	FFFFFF

A summary of the contents of the `routes.txt` *file in the Chicago Transit Authority's feed. Note how 8 of the 134 routes have a color assigned, with 8 unique colors – and 8 routes have a text color assigned, with 2 unique text color values (white and black).* ***View in Dispatcher (https://transit.land/dispatcher/feed-versions/eb0cbe5ab41c9cfde0ebae42471ab5b3f712b008)***

From these summaries, it's possible to quickly spot check for optional columns as well as custom columns that go beyond the spec. This data is also available through the Transitland Datastore API for querying across many feeds simultaneously.

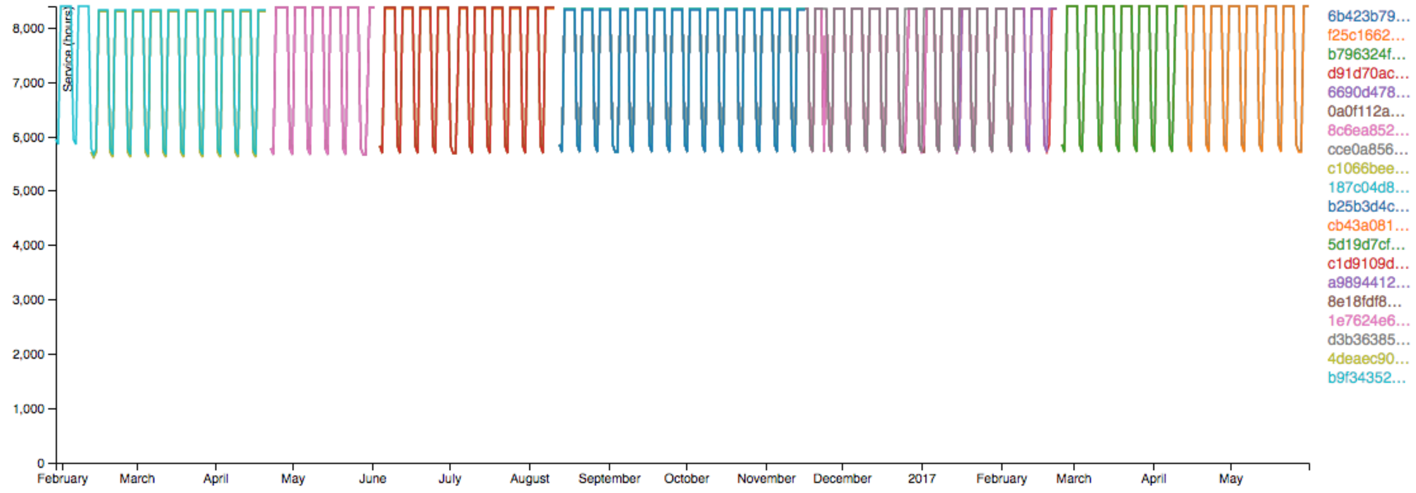
Service-level histograms

Whenever Transitland fetches a feed, it calculates the number of service hours per day. This can be used to generate a histogram showing service coverage for a single feed version:



The seasonal service of the National Park Service Boston Harbor ferries, which peaks on weekends. ***View in Dispatcher (https://transit.land/dispatcher/feed-versions/6e06aeafa0ce200a9da297f20e63bdb7cf065ddf)***

And this can be used to generate histograms showing the extent of service overlap between multiple versions of a feed:

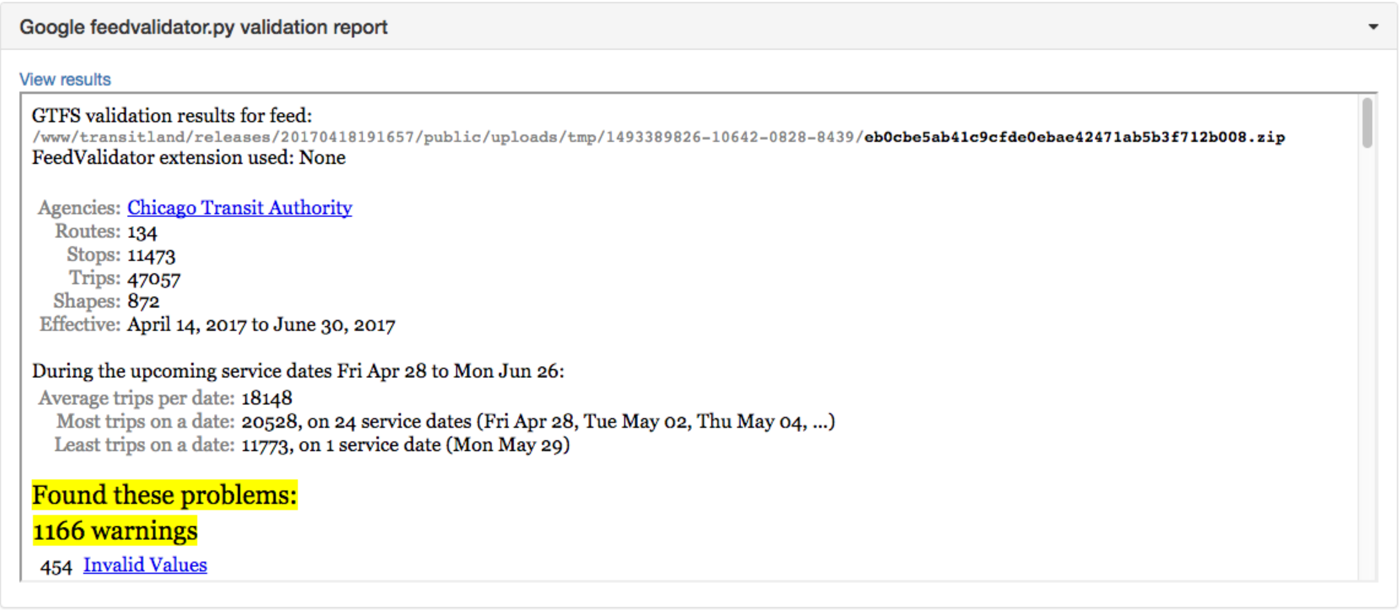


Multiple versions of the San Francisco MTA's feed over the past year. ***View in Dispatcher (https://transit.land/dispatcher/feeds/f-9q8y-sfmta)****, where you can mouse-over to highlight particular feed versions or check/uncheck feed versions to hide them from the graph.*

The service-level counts are available for each day covered by the feed in JSON format, as well, in case you'd like to build your own visualizations or analyses from the Transitland Datastore API.

Google FeedValidator output

When fetching new feed versions, Transitland now runs the **Python-based FeedValidator library from Google (https://github.com/google/transitfeed/wiki/FeedValidator)**. Output can be viewed in Transitland Dispatcher (or fetched as HTML page from the Datastore API):

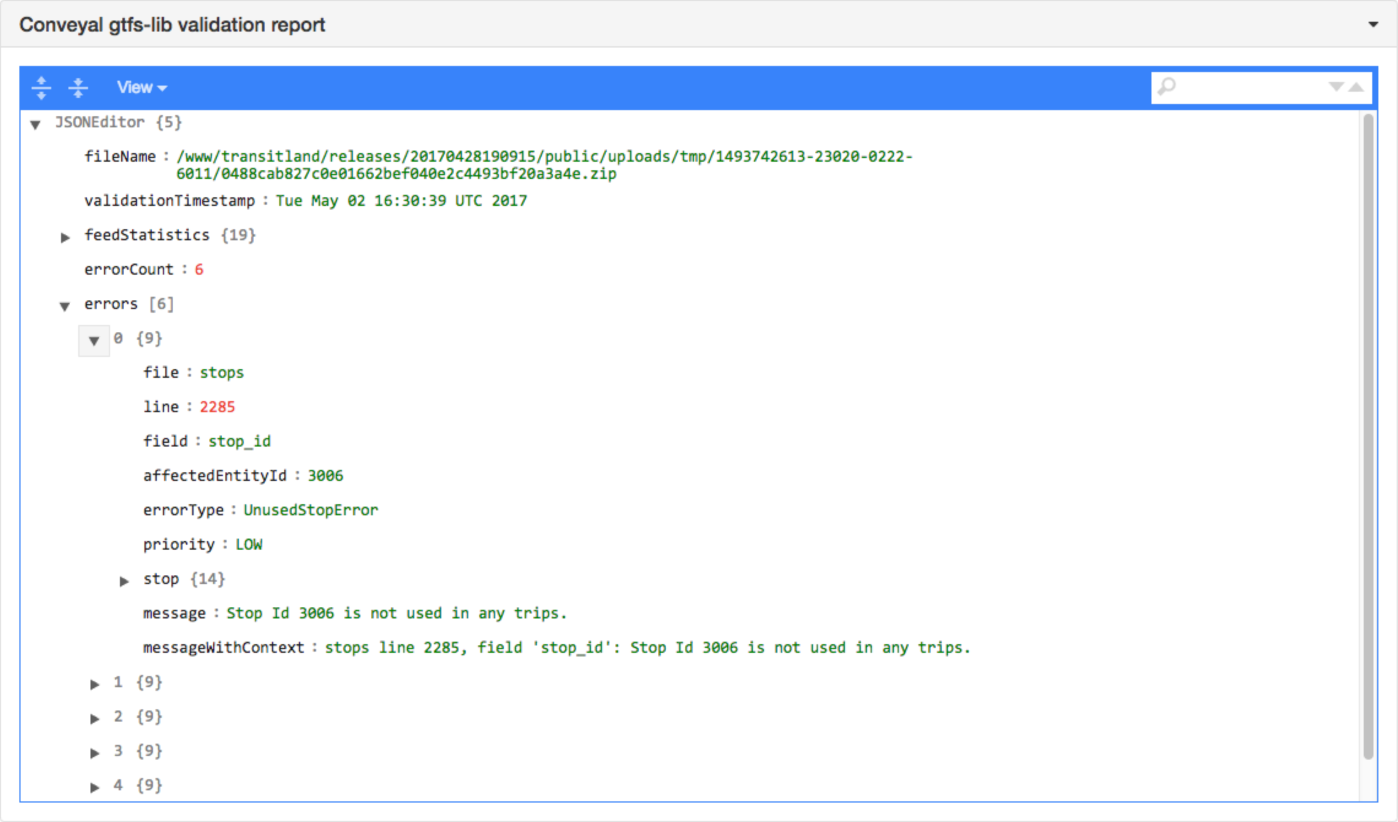


```
GTFS validation results for feed:
/www/transitland/releases/20170418191657/public/uploads/tmp/1493389826-10642-0828-8439/eb0cbe5ab41c9cfde0ebae42471ab5b3f712b008.zip
FeedValidator extension used: None

 Agencies: Chicago Transit Authority
   Routes: 134
    Stops: 11473
    Trips: 47057
   Shapes: 872
Effective: April 14, 2017 to June 30, 2017

During the upcoming service dates Fri Apr 28 to Mon Jun 26:
 Average trips per date: 18148
   Most trips on a date: 20528, on 24 service dates (Fri Apr 28, Tue May 02, Thu May 04, ...)
  Least trips on a date: 11773, on 1 service date (Mon May 29)

Found these problems:
1166 warnings
 454  Invalid Values
```

Google FeedValidator output for a CTA feed version. ***View in Dispatcher (https://transit.land/dispatcher/feed-versions/eb0cbe5ab41c9cfde0ebae42471ab5b3f712b008)***

Conveyal gtfs-lib output

Transitland also runs **Conveyal's (http://conveyal.com/) gtfs-lib (https://github.com/conveyal/gtfs-lib#validation-result)** and stores its validation results for browsing:



```
▼ JSONEditor {5}
    fileName : /www/transitland/releases/20170428190915/public/uploads/tmp/1493742613-23020-0222-
               6011/0488cab827c0e01662bef040e2c4493bf20a3a4e.zip
    validationTimestamp : Tue May 02 16:30:39 UTC 2017
  ▶ feedStatistics {19}
    errorCount : 6
  ▼ errors [6]
    ▼ 0 {9}
        file : stops
        line : 2285
        field : stop_id
        affectedEntityId : 3006
        errorType : UnusedStopError
        priority : LOW
      ▶ stop {14}
        message : Stop Id 3006 is not used in any trips.
        messageWithContext : stops line 2285, field 'stop_id': Stop Id 3006 is not used in any trips.
    ▶ 1 {9}
    ▶ 2 {9}
    ▶ 3 {9}
    ▶ 4 {9}
```

Conveyal gtfs-lib validation output for a Santa Clara County VTA feed version. ***View in Dispatcher (https://transit.land/dispatcher/feed-versions/0488cab827c0e01662bef040e2c4493bf20a3a4e)***

If you have another analysis or validation library that is open-source, we're glad to look into adding it the Transitland feed-fetch process. In the absence of a canonical validation library, we'll run a couple and let users compare output. We've also been working on **Transitland's own approach to "quality checks" and "quality issues." (https://transit.land/news/2017/02/03/introducing-quality-issues.html)**

Don't see your feed? Please help us **add even more feeds to the Transitland Feed Registry (https://transit.land/documentation/feed-registry/add-a-feed.html)** — the platform will automatically run the reports on those feeds as well!

· 24 May 2017 ·

Ian Rees

Ian spends too much time thinking about cities, land use, and open transit data on the Mapzen Mobility team.

Drew Dara-Abrams

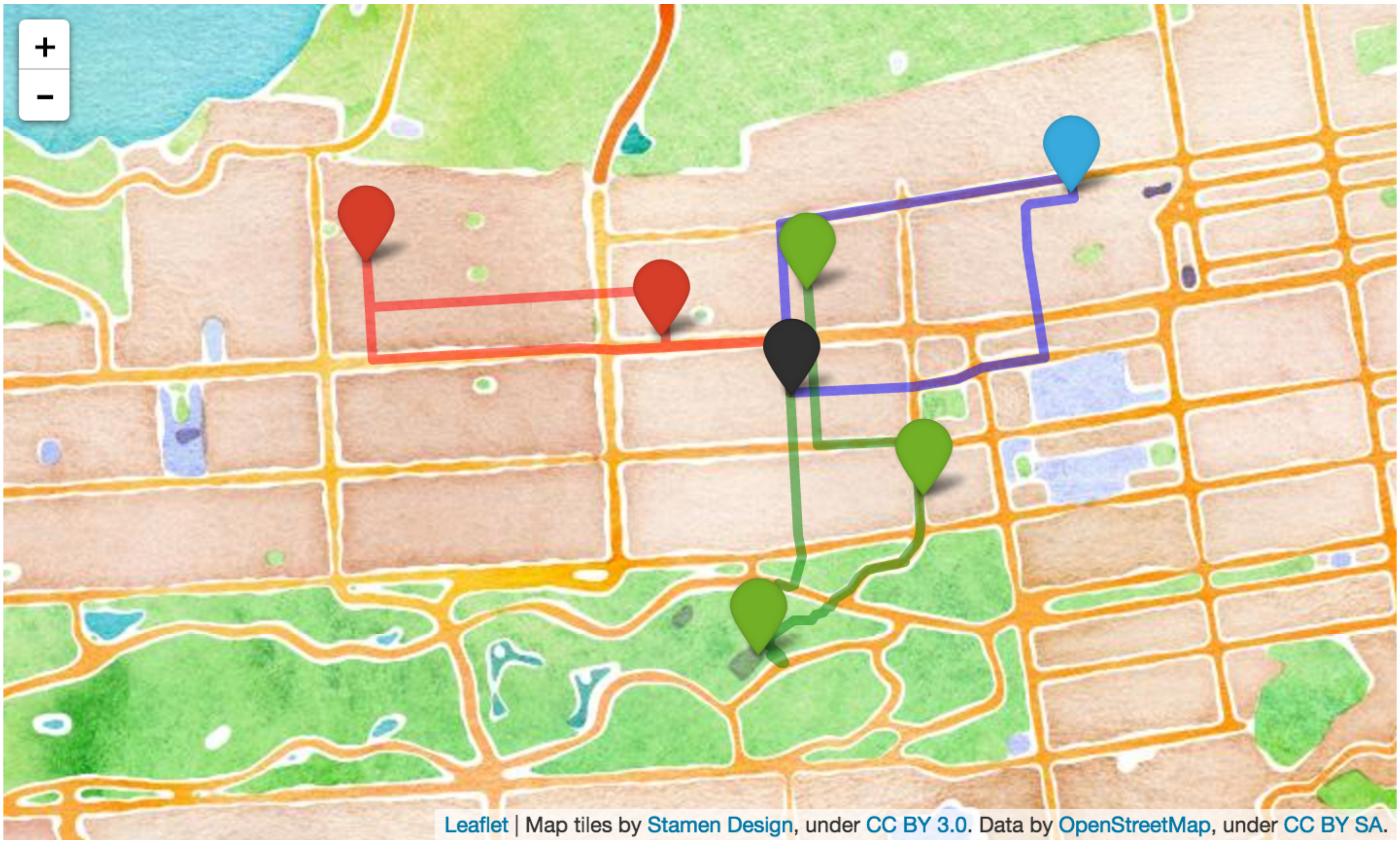
Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

Optimizing Stops

routing **(/tag/routing)** **search** **(/tag/search)**

Andrew Gaidus is an Oakland-based public health GIS analyst. He used Mapzen Optimized Route and Mapzen Search for "traveling salesman" analysis with multiple people and vehicles. Below is an excerpt, and much more detail is available ***on his blog (http://andrewgaidus.com/optimize_stops/)****,* ***Python notebook (https://rawgit.com/agaidus/optimize_stops/master/optimize_stops.html)****, and* ***GitHub repo (https://github.com/agaidus/optimize_stops/)****.*


+
−
Leaflet | Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under CC BY SA.

Output of a travelling salesman optimization function run specifying 3 travelers. In this case, the person with the fewest amount of stops has the longest travel time (52 minutes), and the full set of errands will be made 15 minutes faster than it was with only 2 people.

Traveling Salesman Problem

The traveling salesman problem is a classic optimization problem that seeks to find the most efficient route to connect a given set of points. I recently discovered a set of services built by the open-source mapping company, Mapzen that make this complex problem easy to approximate for a relatively small numbers of stops. Given a set of coordinates, the Mapzen **optimize route service (https://mapzen.com/documentation/mobility/optimized/api-reference/)** uses road network data to produce a time-distance matrix between these points, and then uses an optimization algorithm to determine a route that minimizes total travel time. This can be done for one of three modes of transportation - pedestrian, bicycle, and automobile. They have a great **example (https://mapzen.com/blog/optimized-route/)** with a cool map where they determine the optimal route to visit burrito 'dispensaries' in San Francisco.

This optimization tool can also be used in conjunction with Mapzen's Search Service, which uses open source data to geocode addresses, landmarks, and businesses. Using these two services together is really handy, because it means that one can pass a list of addresses or business names rather than lat / long coordinates.

Building Off the Optimize Route Service

The Mapzen optimize route service takes a set of points and finds the optimal route that a person should take to visit all of these points. However, what if we have multiple "salesmen?" How should the stops be split up between people and in what order should each person visit their stops?

The idea for this was spurred by a project I'm involved with at work, in which we are sending out multiple research assistants to conduct surveys at a dozen or so different sites in Oakland. In this case, it doesn't matter if one person conducts more surveys than the others or who goes to which site - the goal is just to minimize the total time to get them all done.

I decided to test these services in an application that hits closer to home (literally): the optimization of Sunday morning errands between my girlfriend, Celeste, and I. Say we're both starting and ending at our apartment in the Inner Richmond, SF and have 6 different places that we need to stop at. How should Celeste and I split up these errands so that we're finished as quickly as possible?

In the post below, I write a set of functions in Python that call the Mapzen Search and Mapzen Optimize Route API services, making it really easy to determine the most efficient route between a given set of locations. I also extend this function to determine the most efficient way to split the stops among multiple salesmen, and which route each person should take.

Additionally, I use the Python package folium to create leaflet.js slippy maps to display the results of this optimization problem. Folium has a number of built-in tilesets from OpenStreetMap, MapQuest, MapBox, and makes it really easy to build web maps using Python. I find this package to be really useful for data visualization – since I tend to generate data in Python anyway it's nice to be able to do it all within one work environment.

A Few Caveats

The purpose of this blog post is NOT to develop an efficient algorithm to approximate a solution to the vehicle routing problem at any large scale. It is instead to show-case an easy way to access and visualize Mapzen routing services, which I find to be very useful and fun to work with. This post only scratches the surface of the capabilities and range of these routing tools. My use of this service to optimize routes among multiple salesmen addresses a specific aspect of this service that they do not provide, but my approach is more of a 'quick and dirty' demonstration than anything else.

The number of unique ways that stops can be partitioned grows very quickly as the number of stops and salesmen increases, so my approach of testing these unique combinations using Mapzen's optimize route tool will only work for relatively small numbers before exceeding API service limits. For my purposes right now, this is just fine. A better approach would perhaps be to address this problem more "up-stream", using the Mapzen generated Time-Distance Matrix rather than the results from a tool that uses that matrix behind the scenes anyway. However, my approach provides a relatively simple way to get an answer without having to develop my own traveling salesman optimization algorithm.

A Jupyter notebook with all of my code plus more details can be downloaded **here (https://rawgit.com/agaidus/optimize_stops/master/optimize_stops.html)** (or viewed in HTML format **here (http://andrewgaidus.com/images/optimize_stops.html)**). It can also be found **on my GitHub (https://github.com/agaidus/optimize_stops/)**.

· 02 June 2017 ·

Andrew Gaidus

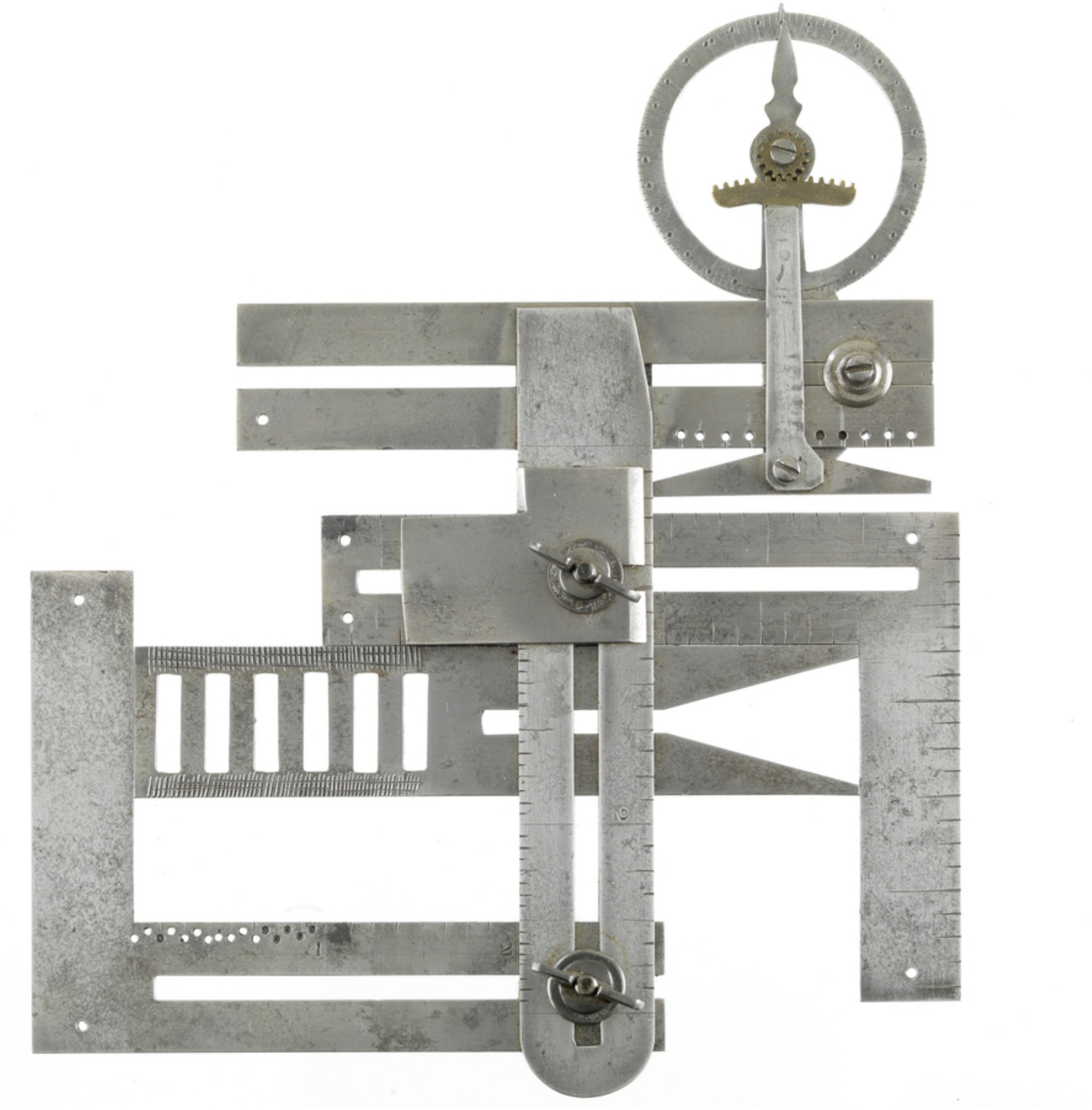
Andrew Gadius is an Oakland-based public health GIS analyst.

© 2017 Mapzen

Everything you wanted to know about Mapzen Search endpoints but were afraid to ask

search (/tag/search)

“Search” and “geocoding” are blanket terms often used to describe any and all types of geospatial search operations. It can be pretty overwhelming when you’re first learning all the terminology, so we’re here to help make sense of all-the-things. We’ll go through what is what, and which of our API “endpoints” will help you find what you need.



First, a little vocabulary.

Words to know

au·to·com·plete

/ˌôdōkəmˈplēt/; synonyms: type-ahead

Anticipating what the user is trying to input and providing meaningful, real-time suggestions: *"It was helpful when* `New York, USA` *popped up as an autocomplete suggestion after I started typing* `new` *."*

ge·o·cod·ing

/ˈjēəˌkōdiNG/; synonyms: forward geocoding, forward search, search

Assigning geographical coordinates to an address, venue, or other location type: *"When geocoding the Mapzen NYC office address, you get the latitude, longitude of 40.744236, -73.990352."*

re·verse ge·o·cod·ing

/rəˈvərs ˈjēəˌkōdiNG/

Determining the nearest address or venue at a given latitude, longitude: *"The app uses reverse geocoding to determine the address where the driver should pick me up.*

struc·tured ge·o·cod·ing

/ˈstrək(t)SHərd ˈjēəˌkōdiNG/

Assigning geographical coordinates to an address, venue, or other location type that has been broken up into its constituent parts: *"Because the user filled out the shipping address form in individual fields, structured geocoding was the easiest way of looking up that location because it was already in its component parts."*

Which API endpoint to use when

Now that we have the vocabulary in our arsenal, let's examine a few different scenarios where geocoding is applicable. Each use case might call for a different variety of spatial search and it's worth describing each in some detail.

You have a single address field that needs geographic coordinates (`/search`)

In this scenario, your data might have customer records, for example, with a single field provided for the address information. The address field might also have a variety of data in it, and is not guaranteed to be formatted in any specific way.

id	name	address	occupation
321	Jane Banks	17 Cherry Tree Lane, London	student
654	Paddington Bear	32 Windsor Gardens, London	bear
987	Sherlock Holmes	221B Baker St, London	private investigator

If your data is free-form like those entries, try forward search with the `/search` endpoint. It requires a single `text` input to be specified, which is set to the individual string for each of your records.

You have an address split into its constituent parts (`/search/structured`)

You might have data that is broken up into columns, where each column represents a part of the address.

id	name	address	city	country	occupation
321	Jane Banks	17 Cherry Tree Lane	London	GBR	student
654	Paddington Bear	32 Windsor Gardens	London	GBR	bear
987	Sherlock Holmes	221B Baker St	London	GBR	private investigator

You'll want to go with structured geocoding (`/search/structured`) for this scenario. If you already know which part of the address corresponds to which field, there is no need to concatenate them all together only to need to break them up again to search. This will help avoid potential errors in both the concatenation and the parsing processes, so let's just save everyone the trouble.

Your columns might vary depending on your database design or input forms used to collect the data, so don't dismay if your columns don't have the same names as the ones in our example. We support a variety of **address parts (https://mapzen.com/documentation/search/structured-geocoding/#structured-geocoding-parameters)** so you can map your columns to each part as needed.

> Tip: If you know more information about your data or would like to limit the search in some way, use filters for that. We allow filtering by country, rectangle, or circle. So for example, notice all the data in the above example is in `GBR`. The more specific you can be in your search requests, the less likely you are to get back an unexpected result.

You have latitude and longitude data and want the corresponding address (`/reverse`)

When you use a mapping app, it's common to click a location to find out the address there. Similarly, when you use your phone to request assistance from the police, for example, your location is often automatically translated to an address where the officers can be routed. In these cases, reverse geocoding is happening behind the scenes.

You use the `/reverse` endpoint when you know the coordinates of your location, and want to learn the address or information about nearest point of interest, such as the name of a business, restaurant, or park.

You have latitude and longitude data and want the administrative area it falls within (`/reverse` with `layers`)

Sometimes, when you click on a map, you want information about the region containing that location. This is a variation of a reverse search, known as coarse reverse geocoding, where you can look up administrative hierarchy information for a given set of coordinates. This can be very useful when you have linked the administrative boundary with other datasets–for example, allowing you to click a house to find out its local government representative or which fire department serves it.

For coarse reverse, use the `/reverse` endpoint and `layers=coarse` or include any of the administrative area types, such as `locality` , `region` , or `country` . If you only want the administrative information, using coarse reverse can improve performance because there are far fewer administrative regions globally than there are individual addresses to query.

You have users typing into a search box or address input field in real time (`/autocomplete` or `/search`)

When you have live users who type things into input fields or search boxes, their activity falls into two buckets, depending on when your app assumes they are ready to search.

The distinction in the results is visible when you consider the behavior with the text of `lond`. Autocomplete for `lond`, returns `london` (**https://mapzen.com/search/explorer/?query=autocomplete&text=lond (https://mapzen.com/search/explorer/?query=autocomplete&text=lond)**) because the user is still typing, but `/search` assumes the user is done entering text and returns the city of `Lond` in Pakistan (**https://mapzen.com/search/explorer/?query=search&text=lond (https://mapzen.com/search/explorer/?query=search&text=lond)**).

…and want to show feedback as they type (`/autocomplete`)

When you are searching for a hotel in a particular city, for example, you often see a list of matching results appear after each character as you type. In your own app, if you want your users to enter text and see real-time results as they type, use `/autocomplete`. This type-ahead functionality helps users find what they are looking for, without requiring them to fully specify their search term. Typically, your users start typing and a drop-down list appears where they can choose the term from the list.

Because `/autocomplete` results in many more individual requests to get a good result, autocomplete has higher free rate limits and lower per-request prices than `/search`.

…and want to search only when they finish typing (`/search`)

If you want the search to occur only when specifically executed, such as when your users press Enter, use `/search`. A search operates under the assumption that the input text is complete. A common design pattern is for an app to use autocomplete for the initial type-ahead behavior, and then switch to a `/search` when the user presses Enter or clicks an entry from the list.

The `/search` endpoint offers some functionality that is not currently available for `/autocomplete`. This includes structured searching and address interpolation for locations where data is incomplete.

(You probably noticed that the examples above point to our new **Mapzen Search Explorer (https://mapzen.com/search/explorer)**. This handy tool not only lets you dig into search endpoints, but lets you adjust parameters and see how they affect results, all on the fly. You can also send the URL to your friends, and they will see the query, its parameters, and the results!)

Image: Patent Model for Carpenter's Combination Tool, 1867, Patent No. 70,547 invented by John Graham, ***via Cooper-Hewitt (https://collection.cooperhewitt.org/objects/35460339/)***

· 08 June 2017 ·

Diana Shkolnikov

Diana is the engineering director of Search@Mapzen and a proponent of mandatory fun, testing, education, and paleo, not necessarily in that order.

Rhonda Glennon

Rhonda is Mapzen's technical publications manager and writes about maps and developer tools.

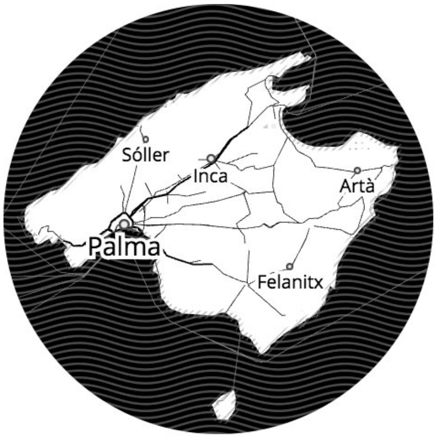
© 2017 Mapzen

Refill Themes - Part 1

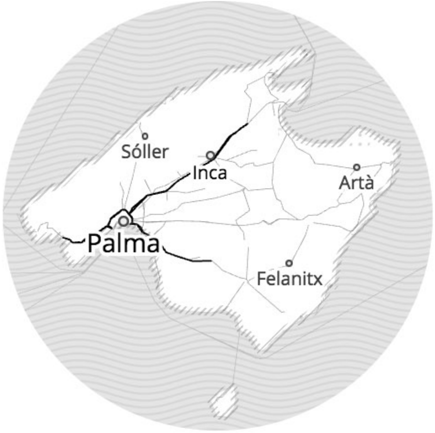
cartography (/tag/cartography) tangram (/tag/tangram)

We’re excited to introduce you to Refill themes! See all the variations below:

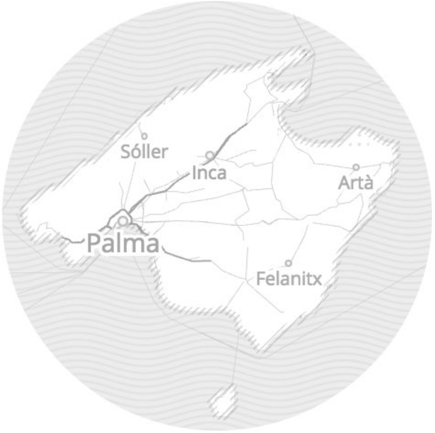
The island of Mallorca


Sóller
Inca
Artà
Palma
Felanitx

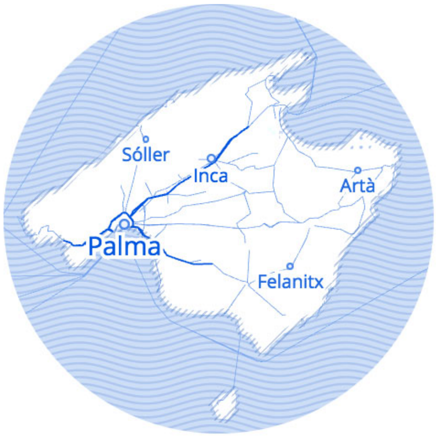
high-contrast


Sóller
Inca
Artà
Palma
Felanitx

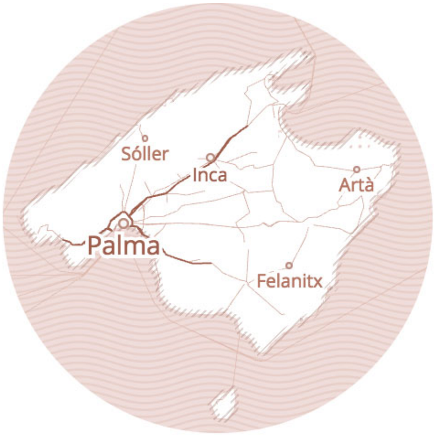
black (default)


Sóller
Inca
Artà
Palma
Felanitx

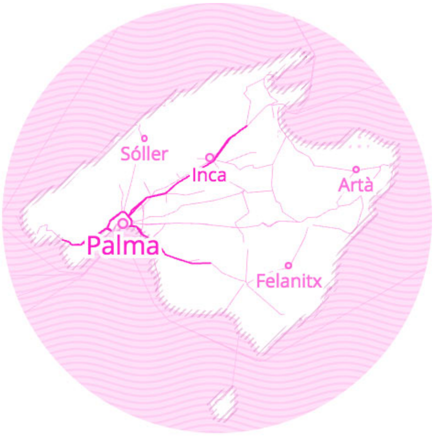
gray


Sóller
Inca
Artà
Palma
Felanitx

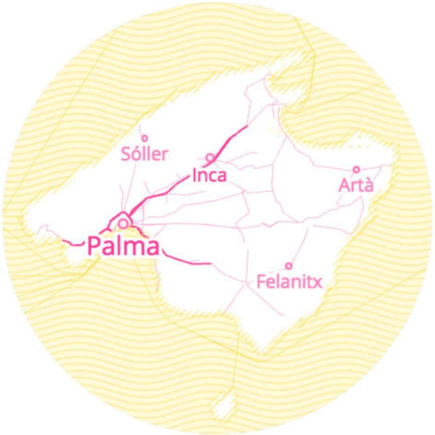
blue


Sóller
Inca
Artà
Palma
Felanitx

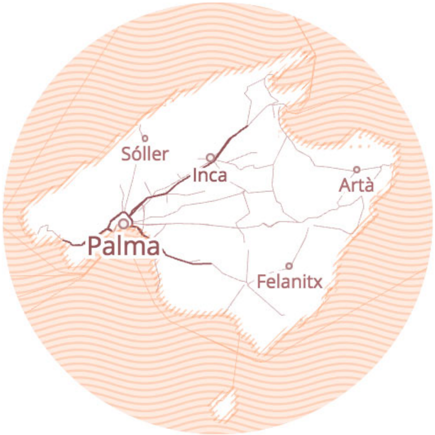
sepia


Sóller
Inca
Artà
Palma
Felanitx

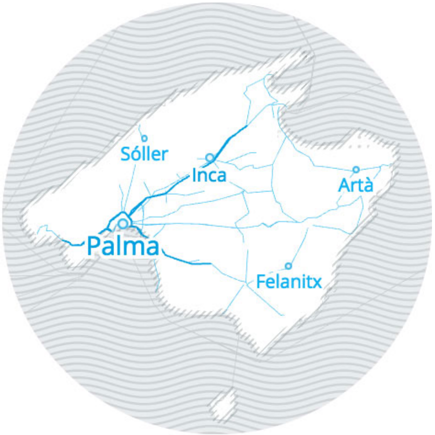
pink


Sóller
Inca
Artà
Palma
Felanitx

pink-yellow


Sóller
Inca
Artà
Palma
Felanitx

brown-orange

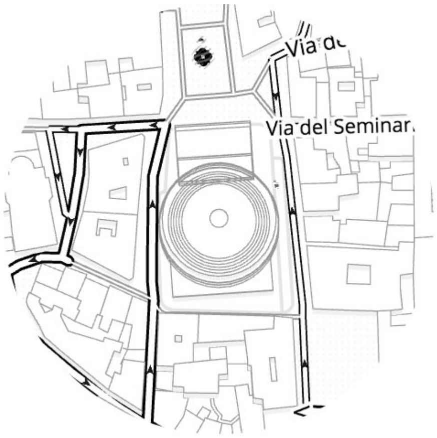

Sóller
Inca
Artà
Palma
Felanitx

blue-gray

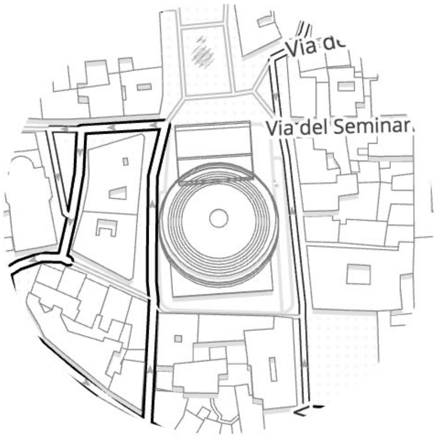
The first row above displays variations on black and white, starting with **high-contrast**, original **black** (Refill's default), and **gray**. High-Contrast is basic and minimal, and great for printing. For data visualization purposes we now have gray and several other muted color themes. On the second row is a single color series in **blue**, **sepia**, and **pink**. The last row displays our new two-color series in **pink-yellow**, **brown-orange**, and **blue-gray**.

We've also introduced three dark three variations in our **Part 2 (http://mapzen.com/blog/refill-themes-part-2)** blog post.

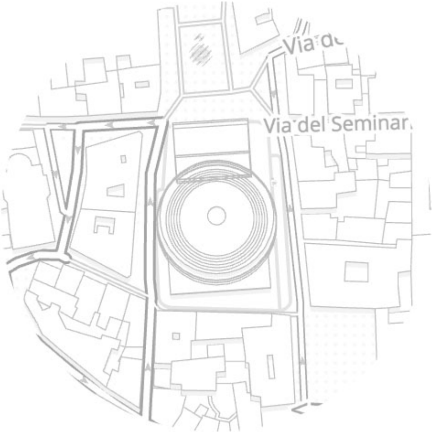
The Pantheon in Rome


Via dc
Via del Seminar

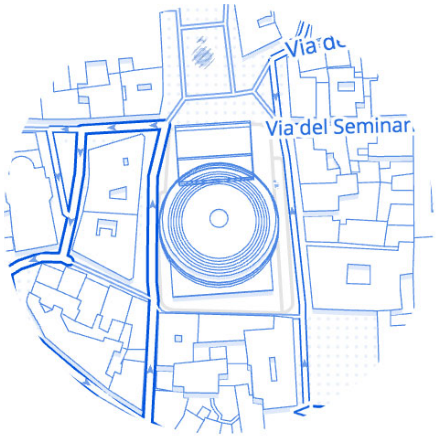
high-contrast


Via dc
Via del Seminar

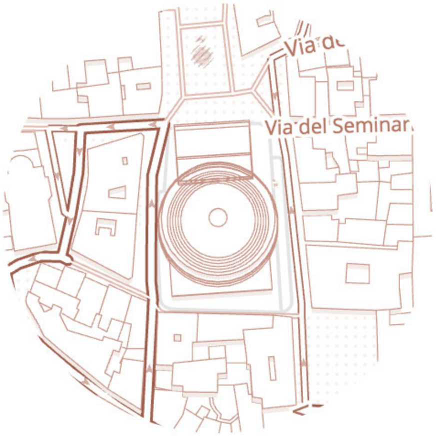
black (default)


Via dc
Via del Seminar

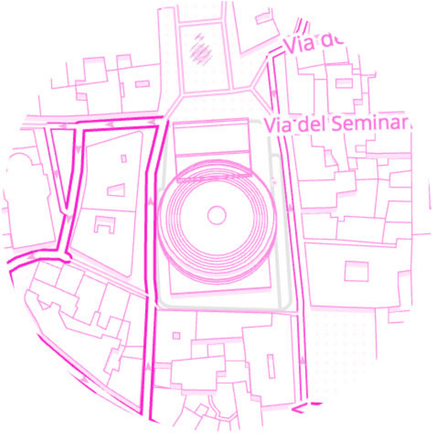
gray


Via dc
Via del Seminar

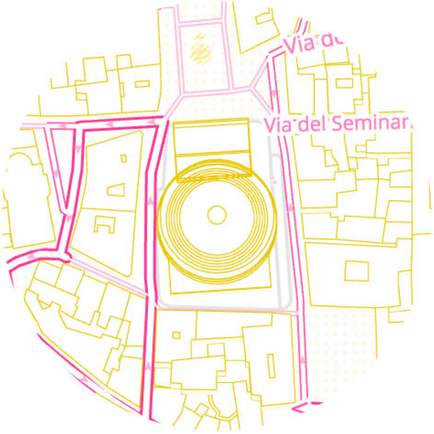
blue


Via dc
Via del Seminar

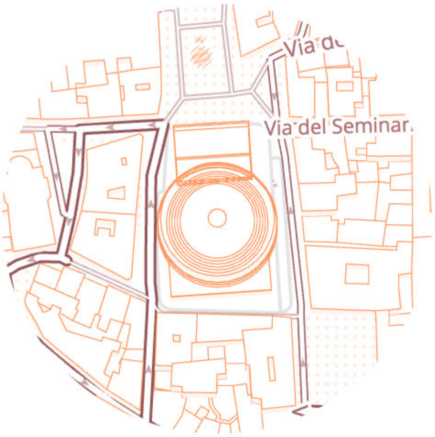
sepia


Via dc
Via del Seminar

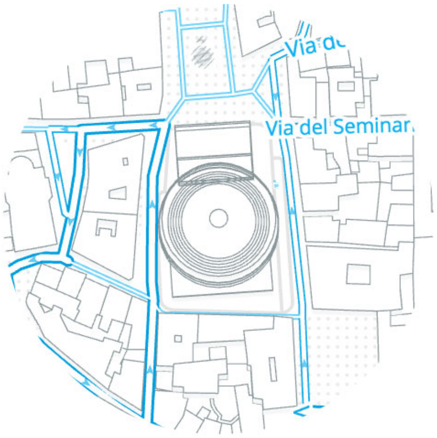
pink


Via dc
Via del Seminar

pink-yellow

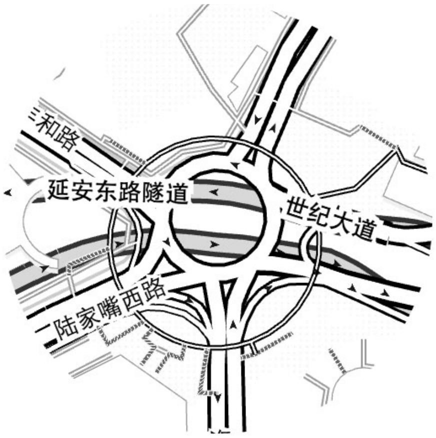

Via dc
Via del Seminar

brown-orange

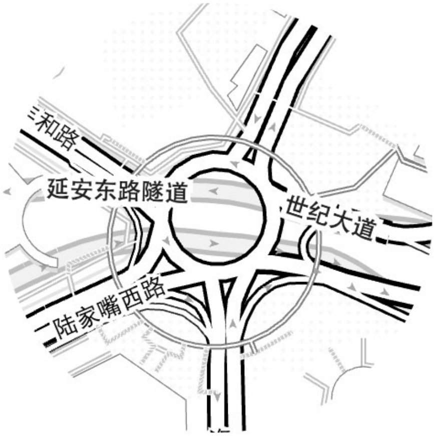

Via dc
Via del Seminar

blue-gray

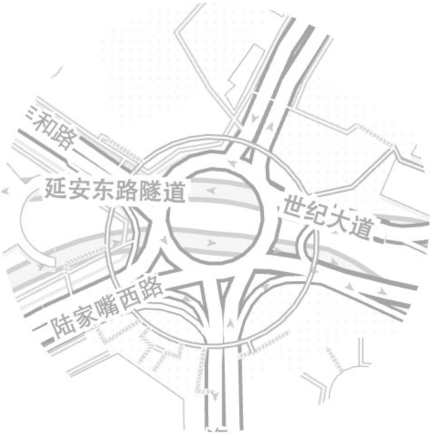
Mingzhu Roundabout in Pudong


和路
延安东路隧道
世纪大道
陆家嘴西路

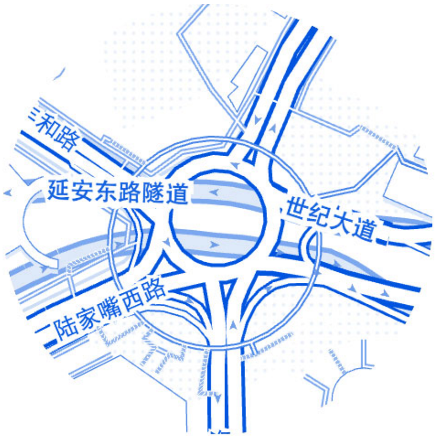
high-contrast


和路
延安东路隧道
世纪大道
陆家嘴西路

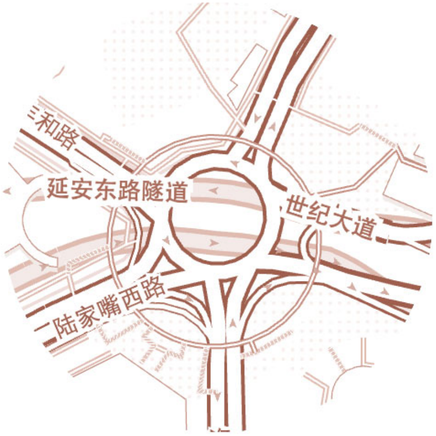
black (default)


和路
延安东路隧道
世纪大道
陆家嘴西路

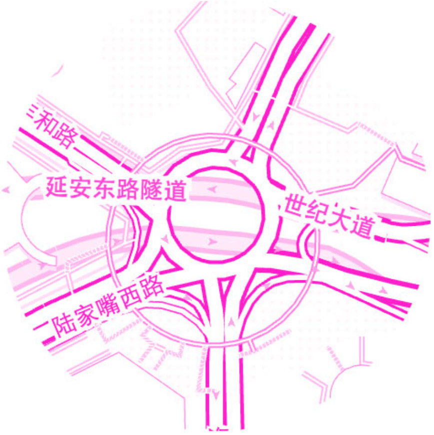
gray


和路
延安东路隧道
世纪大道
陆家嘴西路

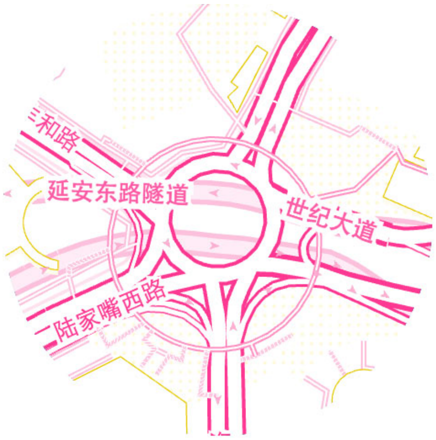
blue


和路
延安东路隧道
世纪大道
陆家嘴西路

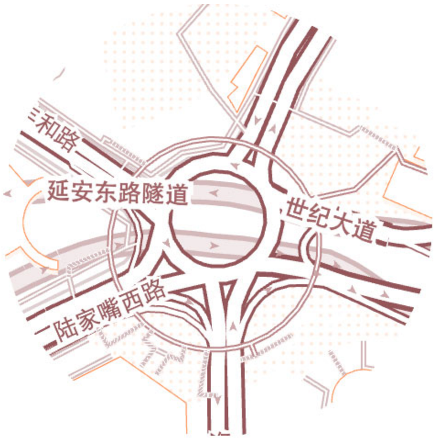
sepia


和路
延安东路隧道
世纪大道
陆家嘴西路

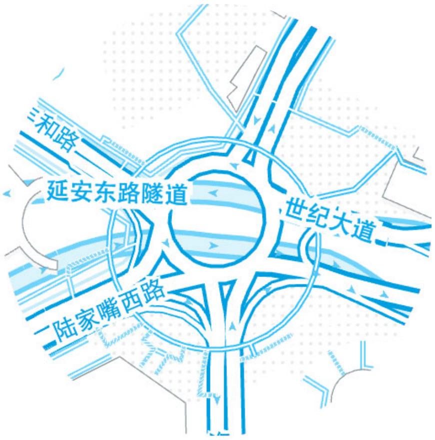
pink


和路
延安东路隧道
世纪大道
陆家嘴西路

pink-yellow

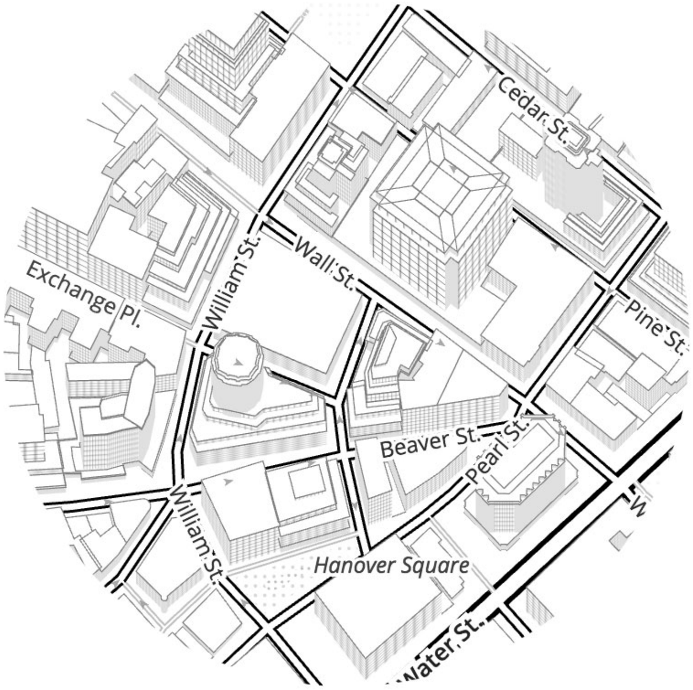

和路
延安东路隧道
世纪大道
陆家嘴西路

brown-orange

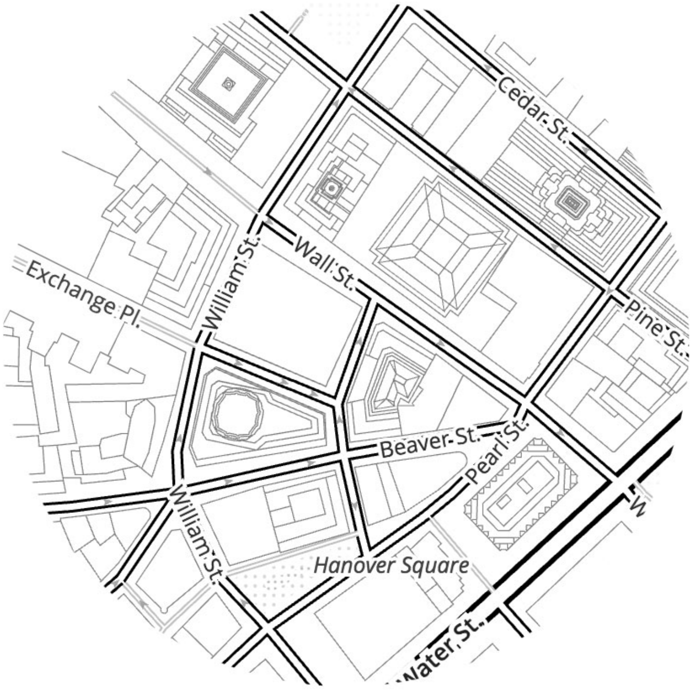

和路
延安东路隧道
世纪大道
陆家嘴西路

blue-gray

Turn 3D buildings ON and OFF

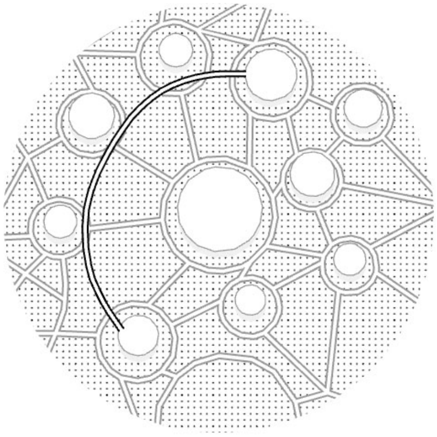

Cedar St.
Wall St.
William St.
Exchange Pl.
Pine St.
Beaver St.
Pearl St.
William St.
Hanover Square
Water St.

3D buildings **YES**

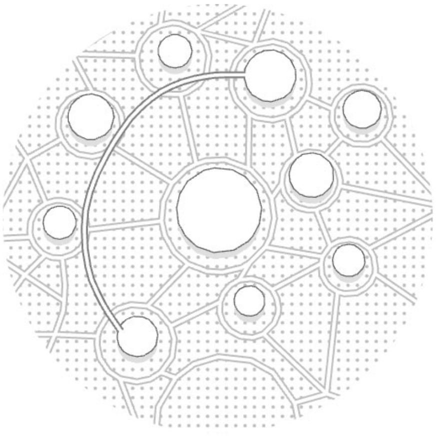

Cedar St.
Wall St.
William St.
Exchange Pl.
Pine St.
Beaver St.
Pearl St.
William St.
Hanover Square
Water St.

3D buildings **NO**

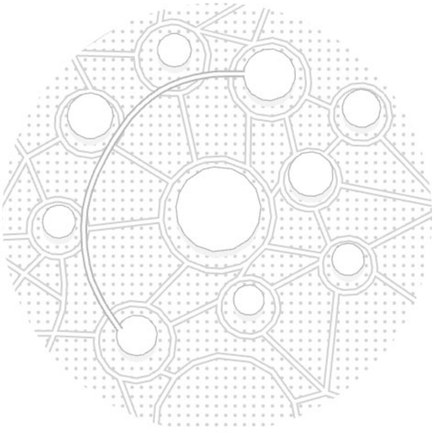
Gardens by the Bay in Singapore



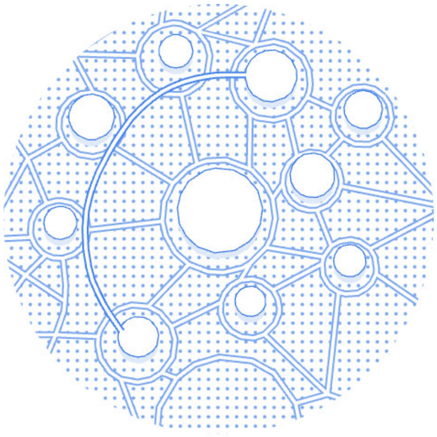
high-contrast



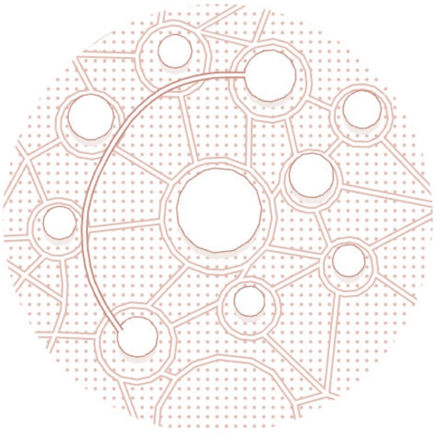
black (default)



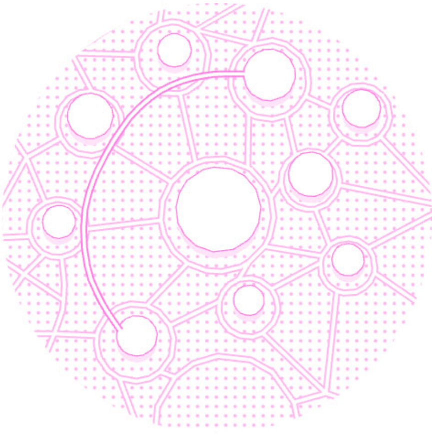
gray



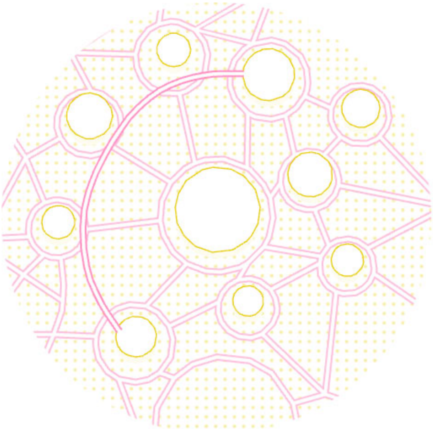
blue



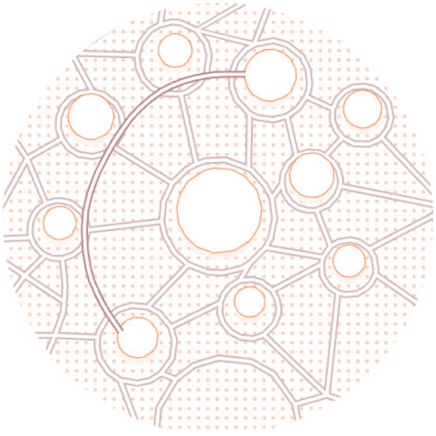
sepia



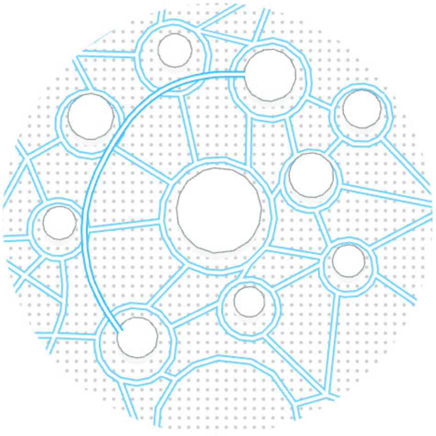
pink



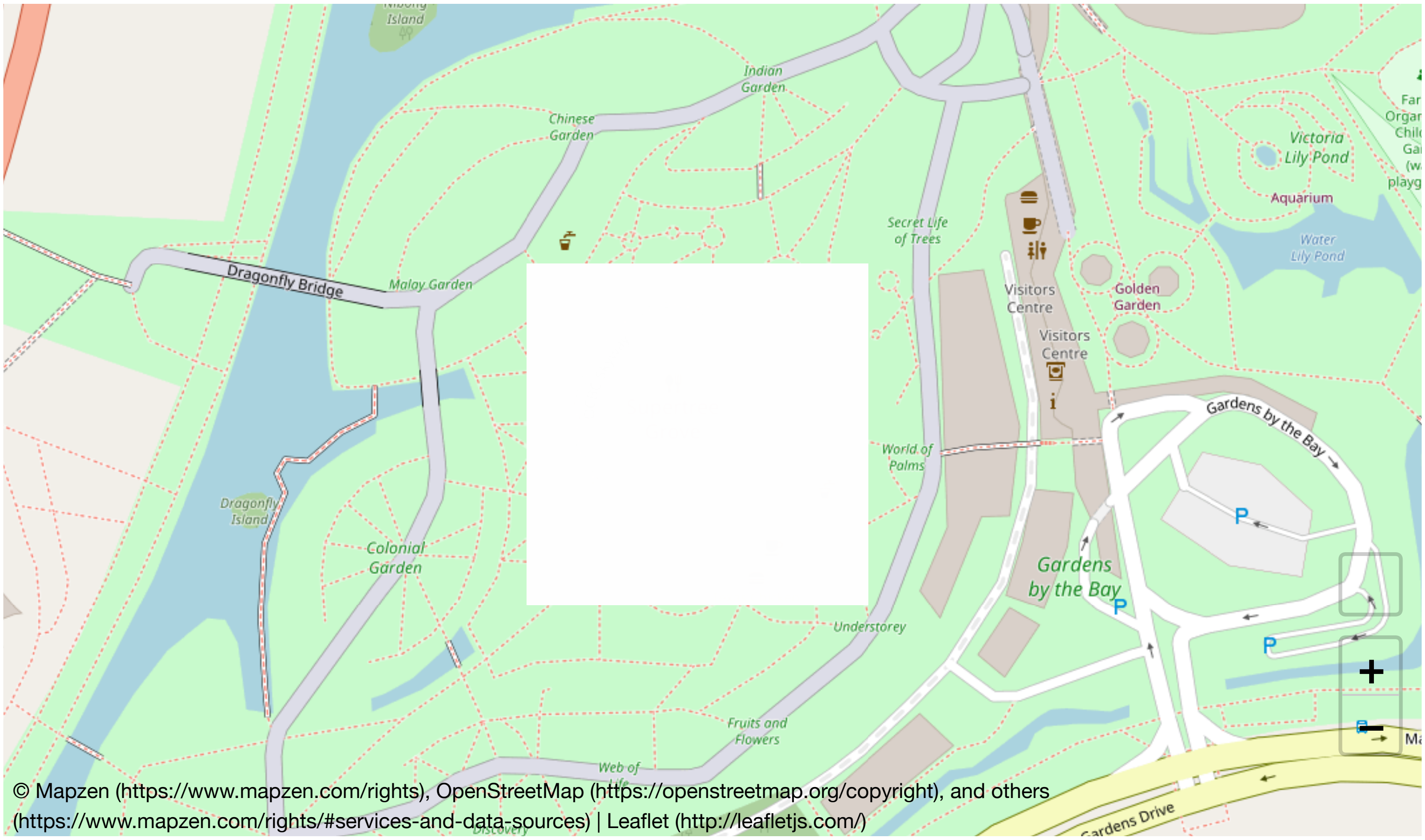
pink-yellow



brown-orange



blue-gray


Island
Indian Garden
Chinese Garden
Dragonfly Bridge
Maloy Garden
Secret Life of Trees
Victoria Lily Pond
Aquarium
Water Lily Pond
Visitors Centre
Visitors Centre
Golden Garden
World of Palms
Gardens by the Bay
Dragonfly Island
Colonial Garden
Gardens by the Bay
P
P
P
Understorey
Fruits and Flowers
Web of Life
Discovery
Gardens Drive
+
−

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

High Contrast

---

*(These maps are interactive! **Open full screen ➶ (https://mapzen.com/products/maps/refill/colors/)**)*

Let there be color

It’s easy to set a theme color after importing Refill in Tangram – just import the basemap and add a few lines! The Cartography **docs (https://mapzen.com/documentation/cartography/themes/#color)** include the following examples.

NOTE: Order matters when importing Tangram YAML files – first import the basemap style, then import themes. You’ll also need to fully qualify the URL for each import as imports are relative to the Tangram scene file’s base URL (not the first import’s base URL).

Set the required **Mapzen API key (https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys)** to access vector tiles and import one of the theme colors:

Example **Tangram YAML** usage:

```
import:
    # basemap style
    - https://mapzen.com/carto/refill-style/8/refill-style.zip
    # recolor basemap style with a theme color
    - https://mapzen.com/carto/refill-style/8/themes/color-pink-yellow.zip
    # other imports to taste...

global:
    sdk_mapzen_api_key: mapzen-xxxxxx
    # are 3d buildings casting too dark a shadow on your map?
    # show 2d building footprints instead (optional)
    sdk_building_extrude: false
```

*NOTE: As Mapzen's basemaps are still in active development we recommend peggging an import to a specific **MAJOR** version, eg:* `7` *, so you enjoy any minor and patch updates but are ensured of stable named scene elements.*

Example **Tangram JS** usage:

```
var layer = Tangram.leafletLayer({
        scene: {
          import: [
            'https://mapzen.com/carto/refill-style/8/refill-style.zip',
            'https://mapzen.com/carto/refill-style/8/themes/color-pink-yellow.zip'],
          global: { 'sdk_mapzen_api_key': 'mapzen-xxxxxx', 'sdk_building_extrude': 'false
        attribution: '<a href="https://mapzen.com/tangram" target="_blank">Tangram</a>, &
    });
```

If you're using **mapzen.js (https://github.com/mapzen/mapzen.js)**:

```
var map = L.Mapzen.map('map', {
  center: [40.8041, -124.1506],
  zoom: 15,
  maxZoom: 20,
  tangramOptions: {
    scene: {
      import: [
        'https://mapzen.com/carto/refill-style/8/refill-style.zip',
        'https://mapzen.com/carto/refill-style/8/themes/color-pink-yellow.zip'],
      global: { 'sdk_mapzen_api_key': 'mapzen-xxxxxx', 'sdk_building_extrude': 'false' }
    } }
});
```

Tangram ES and **Mapzen Android and iOS SDK** support is coming soon.

Happy mapping!

NOTE: This post was updated 2017.07.24 to work with Refill version 8 which namespaces color themes for file imports.

· 13 June 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

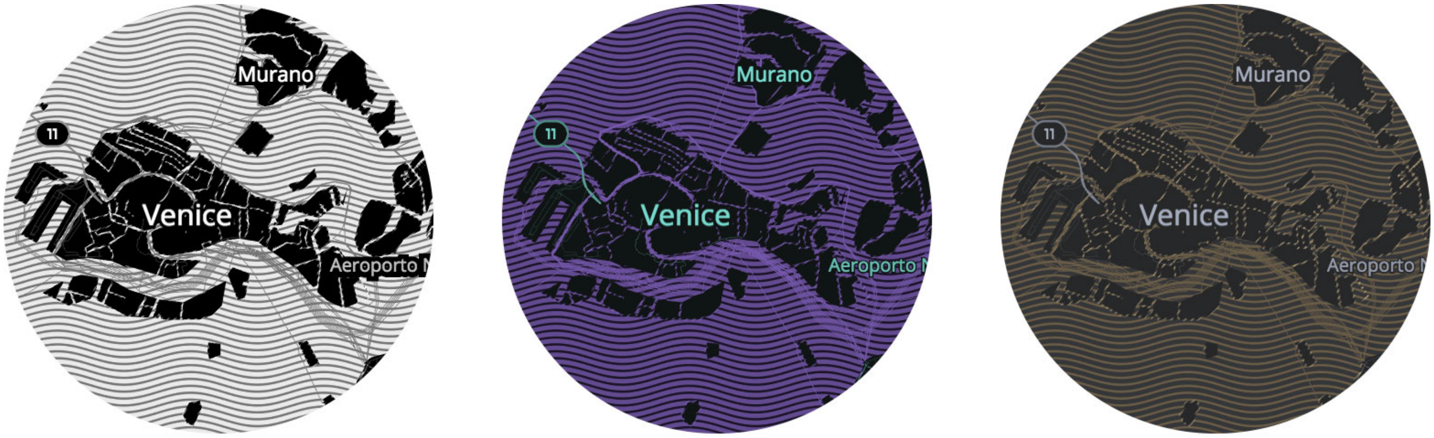
© 2017 Mapzen

Refill Themes - Part 2

cartography (/tag/cartography) tangram (/tag/tangram)

On Tuesday this week, we introduced you to **Refill themes (https://mapzen.com/blog/refill-themes/)**. Today we'd like to continue to the dark color themes in the series. See all three variations below:

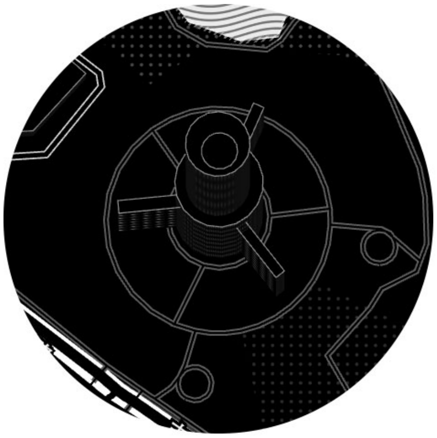
Venice



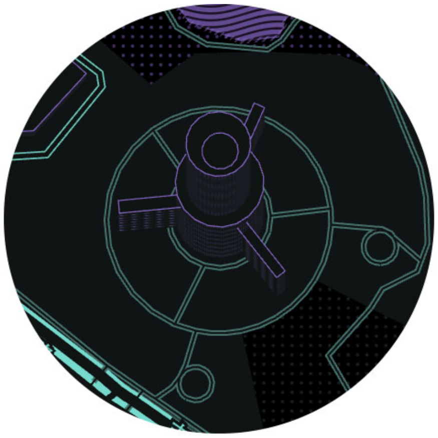
inverted purple-green gray-gold

Dark Refill themes include **inverted**, **purple-green**, and **gray-gold**. **Inverted** is a high-contrast reversed version of the original (black) Refill. **Purple-green** introduces bolder colors to the dark theme. **Gray-gold** is muted in its darks and lights, leaving more range of color play from light to dark for data visualization overlays.

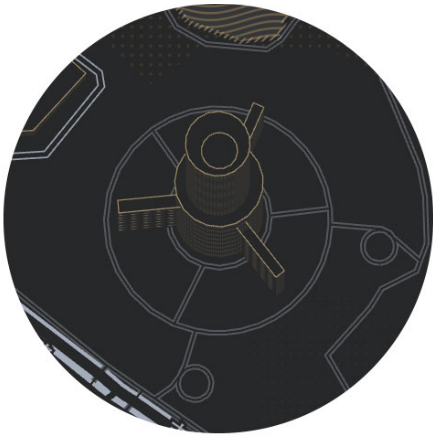
Oriental Pearl Tower in Shanghai



inverted

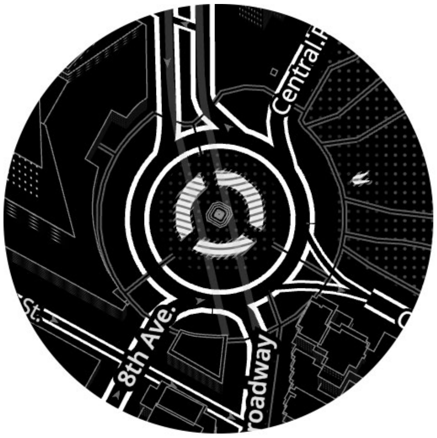


purple-green

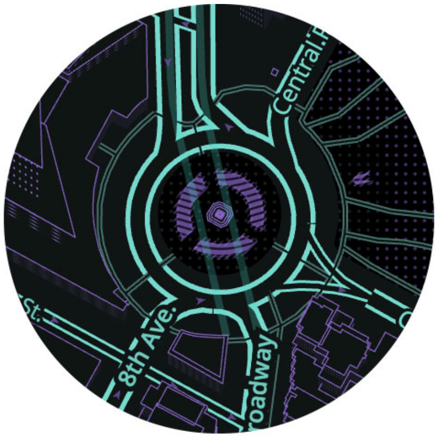


gray-gold

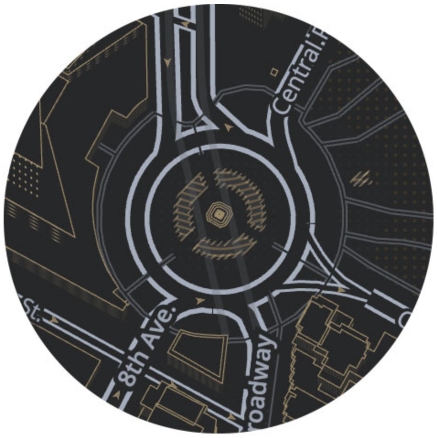
Columbus Circle in New York


Central P
8th Ave.
oadway
St.

inverted

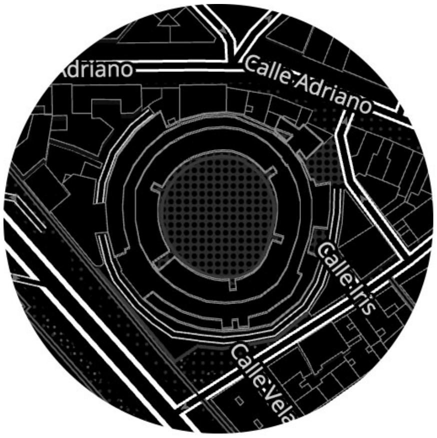

Central P
8th Ave.
oadway
St.

purple-green

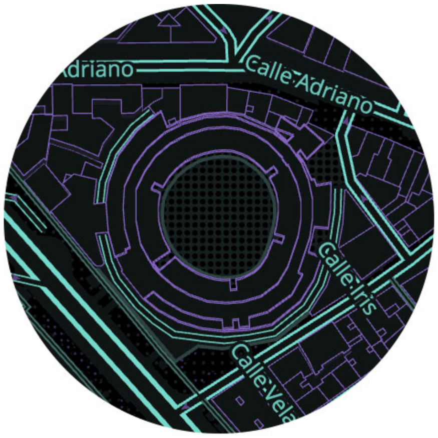

Central P
8th Ave.
oadway
St.

gray-gold

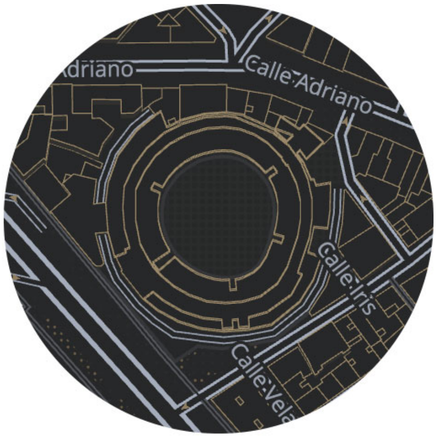
Plaza de toros de la Real Maestranza in Seville


Adriano
Calle Adriano
Calle Iris
Calle Vela

inverted

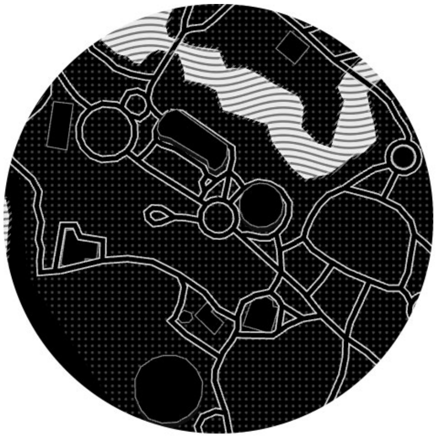

Adriano
Calle Adriano
Calle Iris
Calle Vela

purple-green

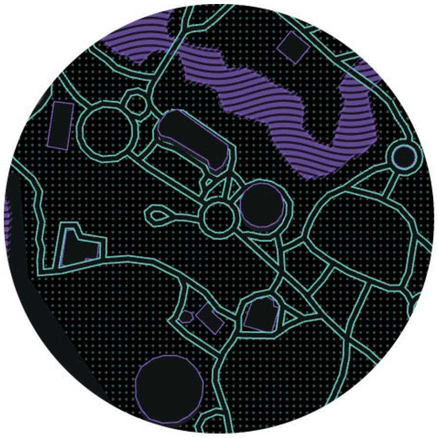

Adriano
Calle Adriano
Calle Iris
Calle Vela

gray-gold

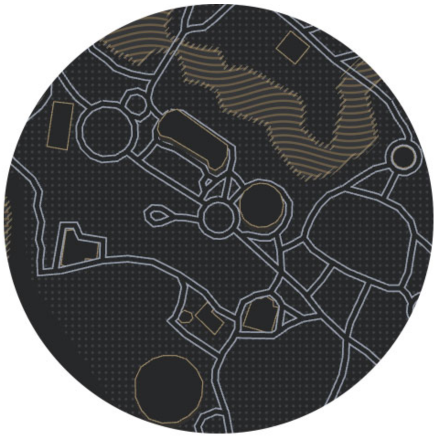
Alipore Zoological Gardens in Kolkata



inverted

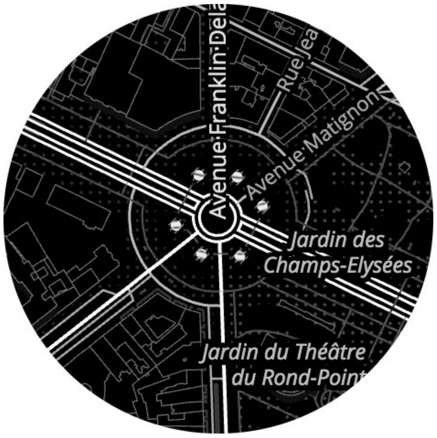


purple-green

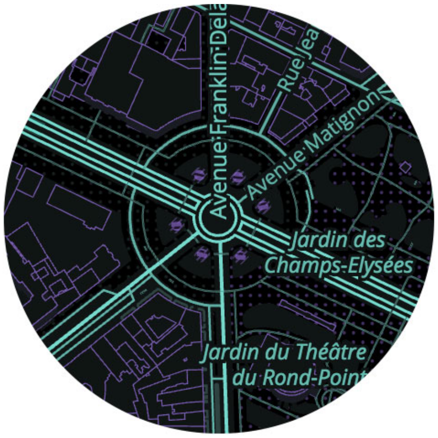


gray-gold

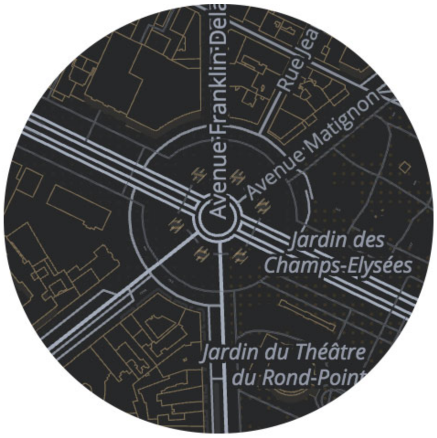
Champs-Élysées in Paris


Avenue Franklin Dela
Rue Jea
Avenue Matignon
Jardin des Champs-Elysées
Jardin du Théâtre du Rond-Point

inverted


Avenue Franklin Dela
Rue Jea
Avenue Matignon
Jardin des Champs-Elysées
Jardin du Théâtre du Rond-Point

purple-green


Avenue Franklin Dela
Rue Jea
Avenue Matignon
Jardin des Champs-Elysées
Jardin du Théâtre du Rond-Point

gray-gold

Rue de Marignan
Spoon
Immeuble Matignon-Mermoz
Rue Jean M
Avenue Matignon
Avenue Matignon
Rond-Point des Champs-Élysées
Avenue Gabriel
Le Gabriel
Théâtre Guignol des Champs-Élysées
Le Laurent
Fontaine du Cirque
Théâtre Marigny
nklin osevelt
Promenade des Champs Elysées
Avenue des Champs-Élysées
Avenue des Champs-Élysées
Promenade des Champs Elysées
Square de Berlin
Champs-Élysées - Clemenceau
Avenue Franklin
Théâtre du Rond-Point
+
−

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

Inverted

(These maps are interactive! ***Open full screen*** ***(https://mapzen.com/products/maps/refill/colors/dark/)*** *)*

Let there be color

It’s easy to set a theme color after importing Refill in Tangram – just import the basemap and add a few lines! The Cartography **docs (https://mapzen.com/documentation/cartography/themes/#color)** include the following examples.

NOTE: Order matters when importing Tangram YAML files – first import the basemap style, then import themes. You’ll also need to fully qualify the URL for each import as imports are relative to the Tangram scene file’s base URL (not the first import’s base URL).

Set the required **Mapzen API key (https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys)** to access vector tiles and import one of the theme colors:

Example **Tangram YAML** usage:

```
import:
    # basemap style
    - https://mapzen.com/carto/refill-style/8/refill-style.zip
    # recolor basemap style with a theme color
    - https://mapzen.com/carto/refill-style/8/themes/color-purple-green.zip
    # other imports to taste...

global:
    sdk_mapzen_api_key: mapzen-xxxxxx
    # are 3d buildings casting too dark a shadow on your map?
    # show 2d building footprints instead (optional)
    sdk_building_extrude: false
```

NOTE: As Mapzen’s basemaps are still in active development we recommend peggging an import to a specific ***MAJOR*** *version, eg:* `7` *, so you enjoy any minor and patch updates but are ensured of stable named scene elements.*

Example **Tangram JS** usage:

```
var layer = Tangram.leafletLayer({
        scene: {
          import: [
            'https://mapzen.com/carto/refill-style/8/refill-style.zip',
            'https://mapzen.com/carto/refill-style/8/themes/color-purple-green.zip'],
          global: { 'sdk_mapzen_api_key': 'mapzen-xxxxxx', 'sdk_building_extrude': 'false
        attribution: '<a href="https://mapzen.com/tangram" target="_blank">Tangram</a>, &
    });
```

If you're using **mapzen.js (https://github.com/mapzen/mapzen.js)**:

```
var map = L.Mapzen.map('map', {
  center: [40.8041, -124.1506],
  zoom: 15,
  maxZoom: 20,
  tangramOptions: {
    scene: {
      import: [
        'https://mapzen.com/carto/refill-style/8/refill-style.zip',
        'https://mapzen.com/carto/refill-style/8/themes/color-purple-green.zip'],
      global: { 'sdk_mapzen_api_key': 'mapzen-xxxxxx', 'sdk_building_extrude': 'false' }
    } }
});
```

Tangram ES and **Mapzen Android and iOS SDK** support is coming soon.

Happy mapping!

NOTE: This post was updated 2017.07.24 to work with Refill version 8 which namespaces color themes for file imports.

· 15 June 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

OSMLR traffic segments for the entire planet

mobility (/tag/mobility)

Since we announced our collaboration with the World Bank and more partners to create the Open Traffic platform (/blog/announcing-open-traffic), we've been busy. We already shared an initial technical preview of the OSMLR linear referencing system (/blog/open-traffic-osmlr-technical-preview), and now we're ready to share another.

OSMLR provides a stable linear-referencing system atop the ever-changing network of roadways in OpenStreetMap. It's used by the Open Traffic platform to associate statistics like speeds and vehicle counts with roadway segments. And it has many more possible uses, beyond just traffic statistics.

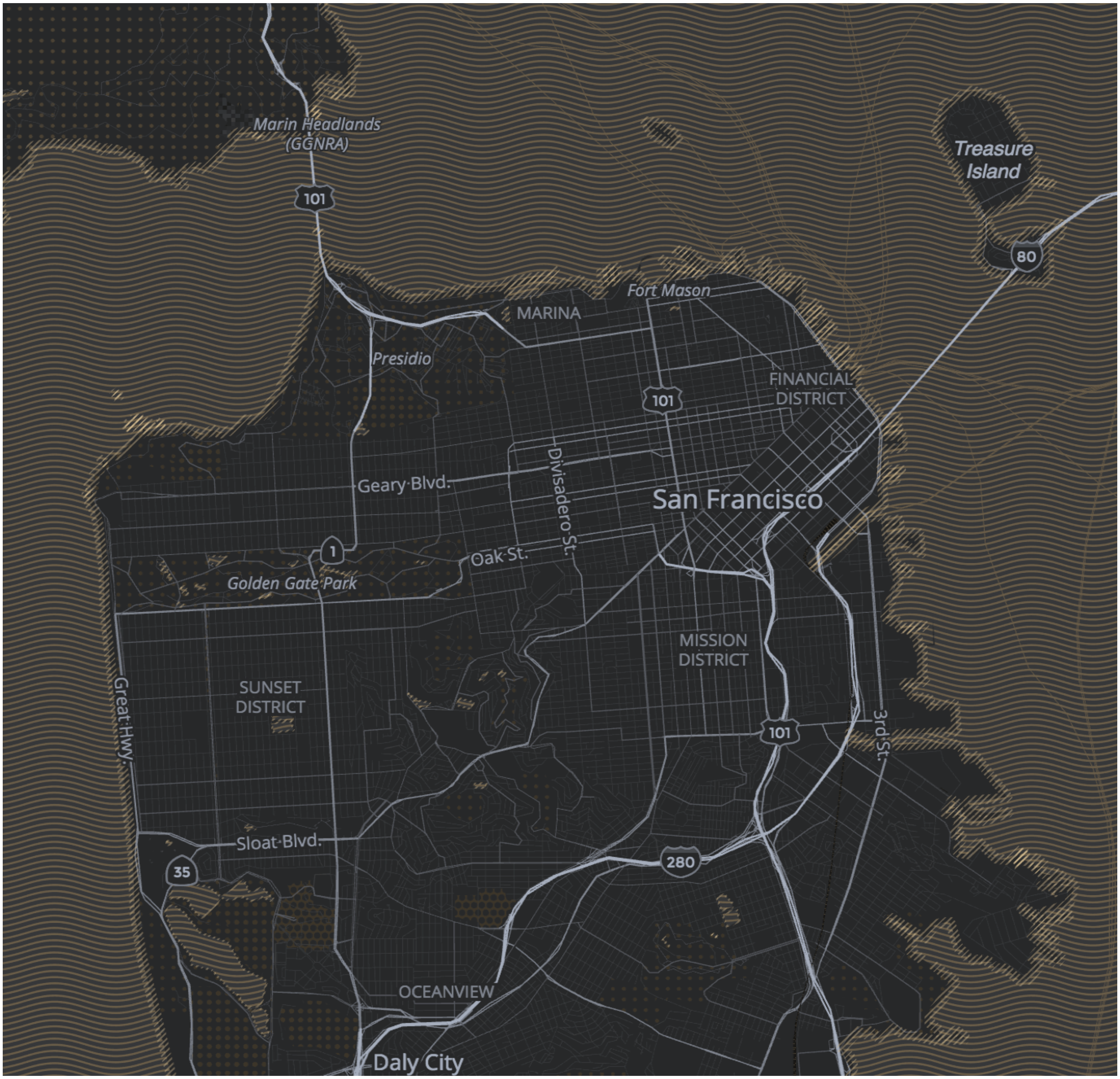
We're now releasing a preview build of OSMLR segment tiles for the entire planet. In this blog post, we explain more about the hierarchical nature of OSMLR tiles, about the criteria for which OSM ways are used to construct OSMLR segments, and how you can download OSMLR segments for testing now.

OSMLR tile hierarchy levels

Similarly to **how the Valhalla stores its routing graph in hierarchical tiles (/blog/valhalla-why_tiles/)**, OSMLR segments are available as geographic tiles at three levels of roadway hierarchy:

- The ***highway level*** (0) includes drivable road segments with OSM highway tags: `motorway` , `motorway_link` , `trunk` , `trunk_link` , `primary` , and `primary_link` .
- The ***arterial level*** (1) includes drivable road segments with OSM highway tags: `secondary` , `secondary_link` , `tertiary` , and `tertiary_link` .
- The ***local level*** (2) includes drivable road segments with OSM highway tags: `unclassified` , `unclassified_link` , `residential` , and `residential_link` .

Here are all three levels of OSMLR segments in San Francisco:


Marin Headlands (GGNRA)
101
Treasure Island
80
Fort Mason
MARINA
Presidio
FINANCIAL DISTRICT
101
Geary Blvd.
Divisadero St.
San Francisco
1
Oak St.
Golden Gate Park
MISSION DISTRICT
SUNSET DISTRICT
Great Hwy.
101
3rd St.
Sloat Blvd.
35
280
OCEANVIEW
Daly City

And here are all three levels of OSMLR segments in Manila, Philippines (where we've previously shown **a prototype of Open Traffic in action) (https://mapzen.github.io/open-traffic-poc-data-demo/)**:

Nepa
Cavite
Solis
Vitas
Andres•Bon
D. Tuazon
Calamba
Dapitan
170
Kalusugan
OBRERO
Velasquez
Rizal Ave.
120
140
SAMPALOG
Cordillera
Santol
140
130
Ipil
Oroquieta
Jacobo Fajardo
NORTH PORT DISTRICT
Rivera
1
Delpan
Juan Luna St.
KIYABO
140
Pureza
Progreso
outh Access Rd.
Arlegui
Malacañang Park
Lubiran
2nd St.
11th St.
Felix Manalo
SOUTH PORT DISTRICT
156
Zamora
J P Rizal
POCO
120
Tejeron
Del Pan
Onyx
Julio Nakpil
President Quirino Ave.
Manila
J.-P.-Rizal
120
2
3
Kamagong

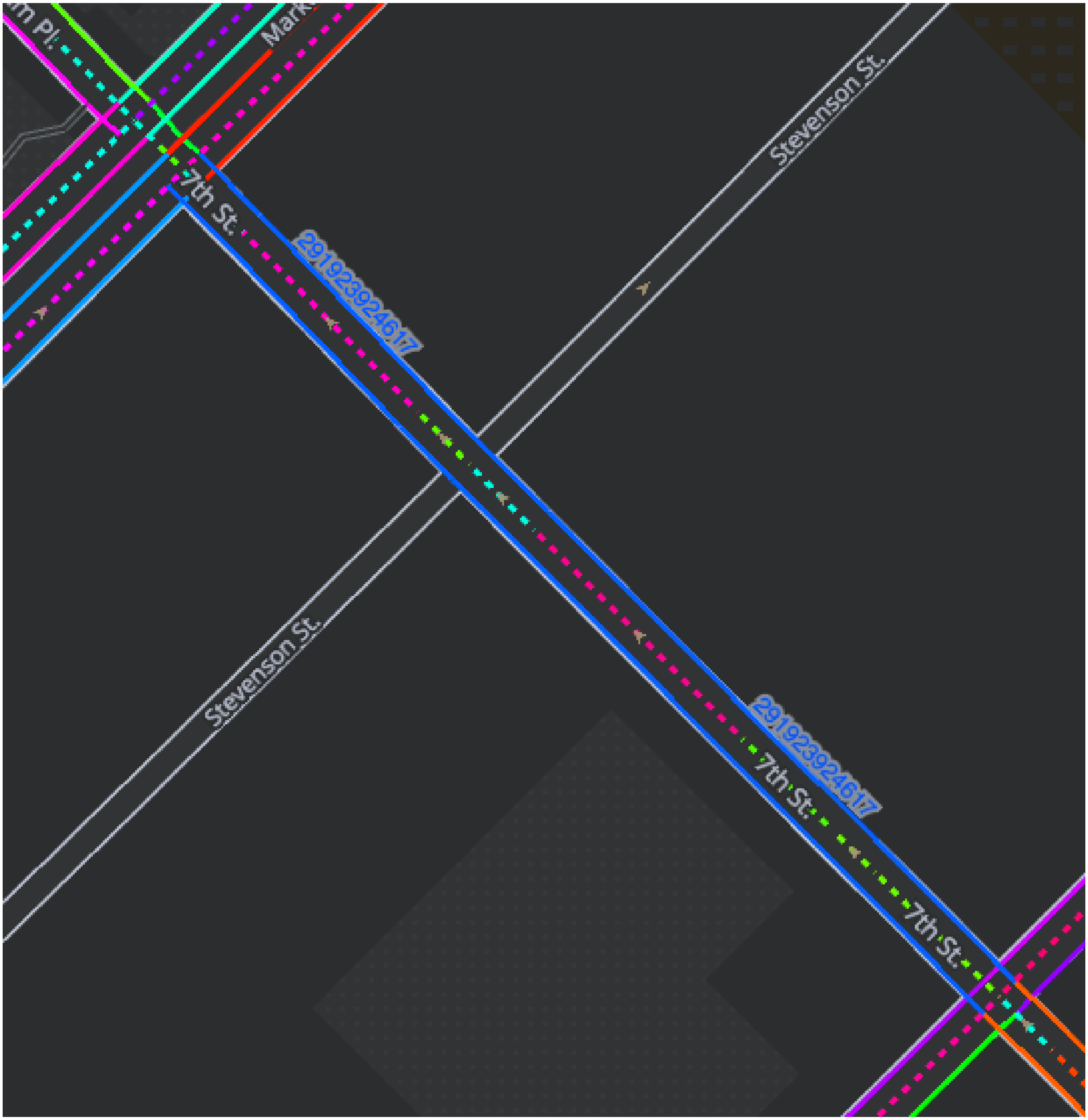
Turning OSM ways into OSMLR segments for traffic applications

OSMLR abstracts away the unnecessary geometric details and boundaries of OpenStreetMap ways. Instead, OSMLR segments summarize the key properties of a meaningful stretch of roadway using “location reference points”, or LRPs. (The concept of LRPs comes from **OpenLR (http://openlr.org/)**.)

We used abstract graphics to illustrate how OSM ways are turned into OSMLR segments in **this previous blog post (/blog/open-traffic-osmlr-technical-preview/)**. With the preview build of OSMLR segments that is now available, we can highlight real-world examples of how OSMLR segments are generated for use in the Open Traffic platform:

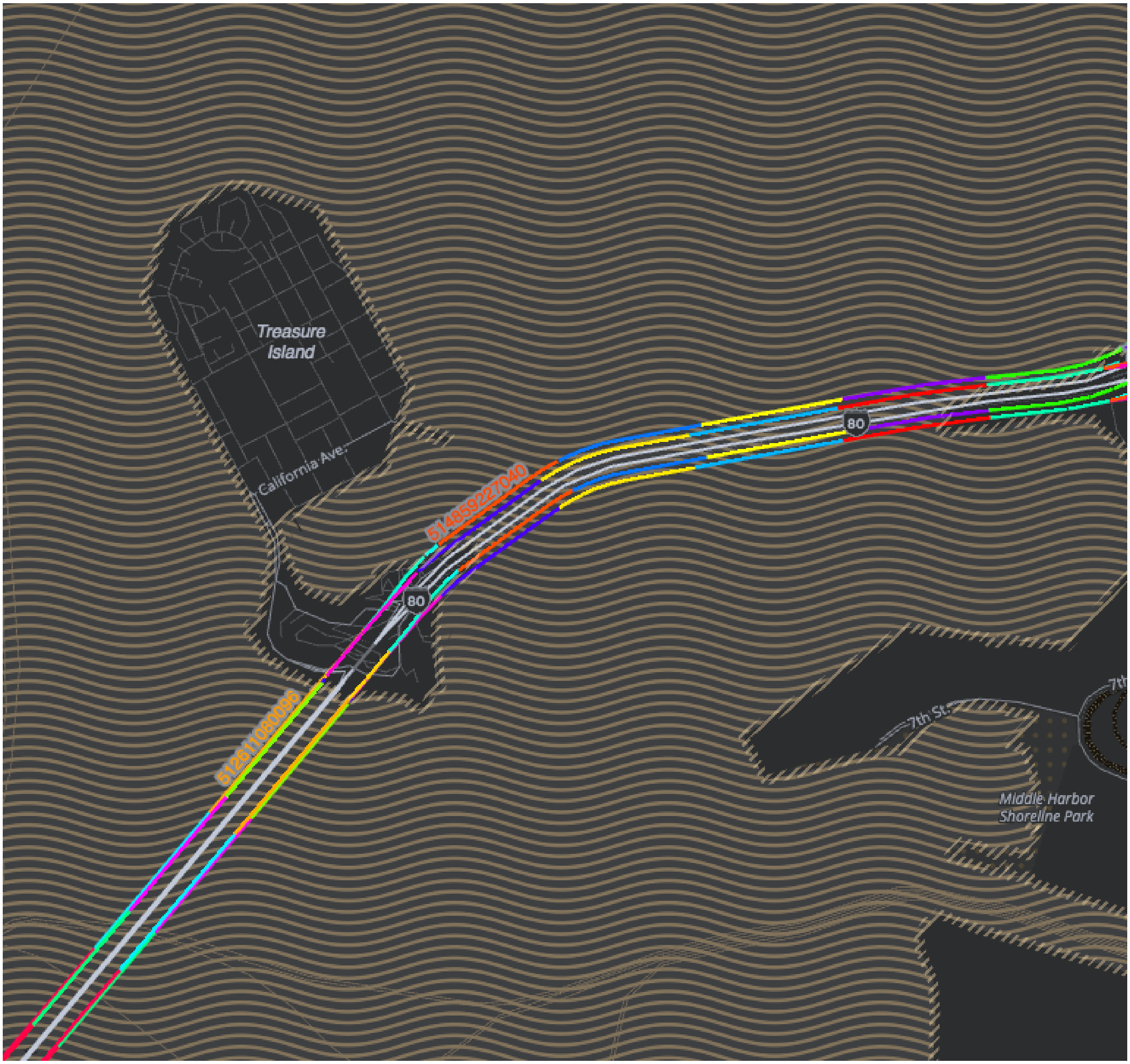
A ***road segment*** represents a section of OSM way(s) between two intersections with other ways.

In some situations, one OSMLR segment may represent multiple OSM ways:


m Pl.
Mark
7th St.
291923924617
Stevenson St.
Stevenson St.
291923924617
7th St.
7th St.

In the above example, one OSMLR segment is drawn in double blue strokes along 7th St. in San Francisco. The OSMLR segment has an ID number of 291923924617. The dotted centerlines indicate OSM ways, with a different color for each OSM way. Note how the OSMLR segment represents the entire stretch of 7th St. between Market St. and Mission St. It joins together multiple OSM ways, which were created in order to provide finer detail for bicycle lane tagging. Those differences are not relevant to OSMLR and its use for Open Traffic. **Open this area on OpenStreetMap.org (http://www.openstreetmap.org/#map=18/37.77977/-122.41255&layers=D)** to explore these OSM ways.

In other situations, one OSMLR segment may represent only a portion of one OSM way:


Treasure Island
California Ave.
514859227040
80
80
512611080096
7th St.
7th
Middle Harbor Shoreline Park

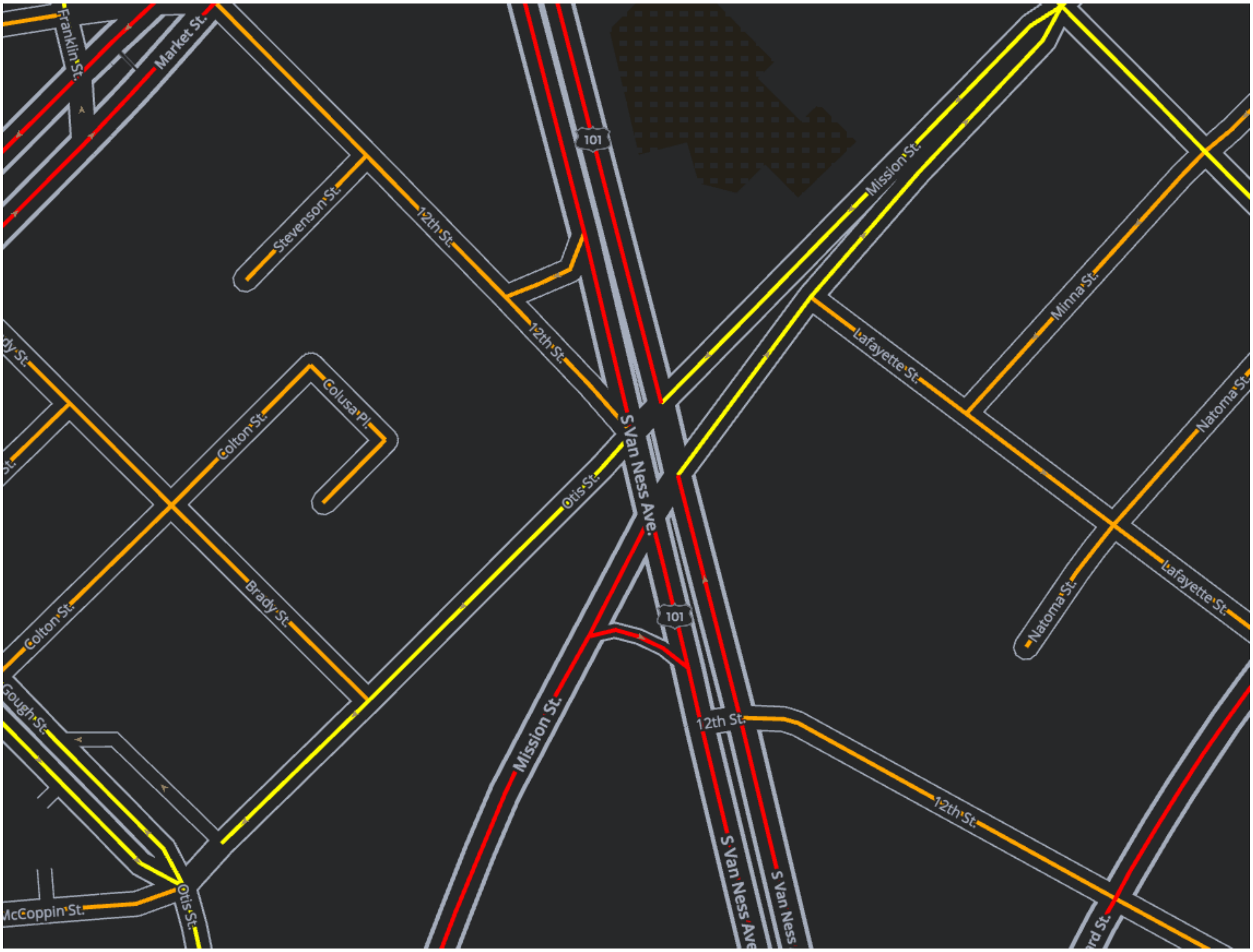
In the above example, the San Francisco-Oakland Bay Bridge is split into multiple OSMLR segments (with a different color for each). The Open Traffic platform will be able to provide different speeds along different lengths of the bridge, so we generate OSMLR segments for each kilometer stretch.

Finally, not every OSM way will be represented by an OSMLR segment at all.

A ***turn channel*** is a short OSM link (highway tag has `_link` at the end) connecting two road segments and is assumed to be at-grade. These cannot connect to motorway and trunk OSM ways cannot fork or connect to other links. Turn channels simplify the path or transition between two roads when turning at an intersection.

An ***internal intersection*** is a short road segment that lies between two one-way, separated road segments. It provides a path or transition between two road segments and generally are not separate carriageways but rather are “internal” to an intersection and are provided to connect different road segments within the intersection.

Here we see an example of an internal intersection in San Francisco where Van Ness Avenue and Mission Street meet. For the purposes of collecting traffic data, it’s unnecessary to have OSMLR segments within the intersection. Instead, we store speed/congestion data by attaching it to two OSMLR segments: the OSMLR segment that the driver enters the intersection using and the OSMLR segment that the driver exits the intersection using.


Franklin St.
Market St.
101
Stevenson St.
12th St.
12th St.
Mission St.
Colusa Pl.
Colton St.
Otis St.
S Van Ness Ave.
Lafayette St.
Minna St.
Natoma St.
Natoma St.
Lafayette St.
Brady St.
Colton St.
Gough St.
101
12th St.
Mission St.
12th St.
S Van Ness Ave
S Van Ness
McCoppin St.
Otis St.

Rules for generating OSMLR segments

Now that you know the basics of where OSMLR segments are defined, let's review some more specific rules we use for generating OSMLR:

- The road segment must be drivable by automobile or other non-emergency vehicle types (bus, taxi, truck, or high-occupancy vehicle).
- A road segment which allows travel in both directions is assigned two OSMLR segments: one in each direction.
- Road segments on all hierarchies are merged or connected across intersections with only non-drivable road segments.
- Road segments on the highway and arterial levels are merged across intersections with only local hierarchy road segments.
- Road segments on the local level include drivable road segments with the exception of service roads (includes parking aisles, driveways, and other minor roads that connect to services).

- OSMLR segments are not generated for turn channels, roundabouts, and internal intersection road segments (as we illustrated above).
- OSMLR segments longer than 1 kilometer are split into multiple segments such that all OSMLR segments are less than 1 kilometer in length. (See the above example of the San Francisco-Oakland Bay Bridge.)

Merging or connecting road segments allows OSMLR definitions to be longer on average and to better average speed along road segments. Creating average speeds or traversal times along extremely short segments can be very inaccurate, especially when the interval between sampled locations (GPS readings) becomes larger. Including a large number of very short segments also creates much more storage and processing demand. However, very long segments do not permit localization of congestion points so a maximum length of one kilometer was settled on.

We plan to generate statistics on “intersection transition” times along with average speeds along an OSMLR segments. This is our rationale for not including turn channels, roundabouts, and internal intersection road segments. These are generally very short road segments and provide a “transition path” between two other road segments. These road segments can be thought of as connecting or transition segments. Since we plan to model the transition times from one OSMLR segment to another we felt that there was no need to form OSMLR definitions for these connecting road segments, we simply merge across them when performing map-matching of GPS traces to OSMLR segments.

Planet-wide OSMLR

We’ve used the the **OSMLR generator application (https://github.com/opentraffic/osmlr)** to cut OSMLR segments for the entire planet at all three levels of the hierarchy. Here are statistics about the process:

- OSMLR segments in total: 216,819,246
- OSMLR segments under 20 meters in length: 10,303,995
- average length of OSMLR segments: 79.24 meters
- maximum number of OSMLR segments within a single geographic tile: 344,209
- average number of OSMLR segments per tile: 1,528
- total tiles to cover the planet: 141,888

Not all applications require residential and unclassified road segments (the local level, 2, of the hierarchy). Some benefits of only using OSMLR for the highway (0) and arterial (1) levels include:

- Large reduction in total number of OSMLR segments (~37 million compared to ~217 million)
- Large reduction in number of tiles (files) needed (~13.5 thousand compared to ~141 thousand)
- Longer average segment length (462 meters compared to 79 meters) and many less short segments

OSMLR Update and Migration Process

Now that we've successfully created OSMLR segments for the entire planet, the next question will be how we update the segments as the road network in OpenStreetMap changes over time.

As OSM edits are made, the vast majority of OSMLR segments remain unchanged. Even most OSMLR segments where the base road segments change will still be valid as the shape and even the endpoints (intersections) can vary slightly and the OSMLR segment remains valid. (See **this previous blog post (/blog/open-traffic-osmlr-technical-preview/#how-does-osmlr-solve-the-problem?)** to learn more about how OSMLR uses location reference points to represent the key properties of a segment, without depending upon the exact geometry of a segment.)

Certain types of changes to OSM will invalidate an OSMLR segment:

- the addition of a new highway or arterial road segment that intersects in the middle of an existing OSMLR segments
- changes where the endpoints of a road segment change above some threshold (to be determined)
- changes where the length of a road segment changes above some threshold (also TBD)

These “breaking” changes will require deprecating the existing OSMLR segment and adding new ones. We are working on a means to mark the lineage and change history of OSMLR segments, so this information will be preserved and available in OSMLR tile files.

We have done some initial tests that show that after six weeks, OSMLR segments still correctly associate to over 99.5% of the underlying OSM road segments. More investigation is needed to see how OSMLR segments “degrade” over time and how frequently they should be updated. We need to balance the cost of updating OSMLR segments (and consider how to update any data associated to the segments) against the growing error rate in associating OSMLR back to OSM road segments.

Read more about considerations for the update/migration process, and track our progress, on **this GitHub issue (https://github.com/opentraffic/osmlr/issues/7)**.

How to Download Preview OSMLR Segments Now

We have created a script that gives you the ability to fetch a subset of OSMLR GeoJSON or protocol buffer tiles.

This script is located within the `py` directory under the **Open Traffic OSMLR repository (https://github.com/opentraffic/osmlr/tree/master/py)**. The script uses a bounding box to determine the list of tiles that intersect with the bounding box.

For example, if you would like to download OSMLR tiles for New York City in protocol buffer format you would execute the following command:

```
./download_tiles.sh -74.251961,40.512764,-73.755405,40.903125 https://s3.amazonaws.com/os
```

And the GeoJSON tiles can be obtained via:

```
./download_tiles.sh -74.251961,40.512764,-73.755405,40.903125 https://s3.amazonaws.com/os
```

Note that this set of OSMLR segment tiles is considered a preview build. In the months ahead, we may make “breaking” changes to the ways in which OSMLR segments are generated and IDs are assigned.

So, please don’t yet depend on the stability of particular OSMLR segment IDs—but please do use this global tile set to begin exploring the possibilities for integrating OSMLR segments into your applications and analyses.

As we release new builds of OSMLR tile files, you’ll be able to view **listing of the latest tile files on S3 (https://s3.amazonaws.com/osmlr-tiles/listing.html)**.

More in the Months Ahead

We’ll be following up in months ahead with another blog post to announce the OSMLR update/migration process. In the meantime, we hope you find this global set of OSMLR segment tiles interesting to explore and consider integrating into your own applications that need stable roadway identifiers.

· 26 June 2017 ·

David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

Tackling Space and Time in Who's On First

whosonfirst (/tag/whosonfirst) data (/tag/data)

Let’s start with Yugoslavia

Creating historical records in the **Who’s On First (https://whosonfirst.mapzen.com/)** gazetteer for Yugoslavia is an excellent test case in helping us answer the following question:

How can Who’s On First catalogue and preserve records for Yugoslavia, the former Soviet republics, and everything in between?

Introducing EDTF

The Library of Congress developed the **Extended Date/Time Format (EDTF) (https://www.loc.gov/standards/datetime/pre-submission.html)**, and is in the process of adding it as an extension to **ISO 8601 (http://dotat.at/tmp/ISO_8601-2004_E.pdf)**. This date/time string is used to express the concept of a fixed or known date (`YYYY-MM-DD` , `YYYY-MM` or `YYYY`), as well as other semantic qualifiers like *“approximately the year 2017”* (`YYYY~`) and *“approximately 2003, but that is uncertain”* (`2003?~`).

This format is perfectly suited to tracking historical administrative records, specifically “places” in **dispute and contention (https://whosonfirst.mapzen.com/spelunker/placetypes/disputed/#4/-1.32/112.73)**. Being able to catalogue a place with as many moving pieces as Yugoslavia allows us to use “soft” beginning and end dates.

The how

Who’s On First uses this format for several **properties (https://github.com/whosonfirst/whosonfirst-properties/blob/master/properties/edtf.md)**, including:

edtf:inception:

> *Indicates the date when a place was first created or established.*

edtf:cessation:

> *Indicates when a place stopped being a "going concern". The semantics for something ceasing may vary from placetype to placetype. For example, a venue may cease operations or a country may split in to multiple countries.*

edtf:deprecated:

> *Indicates the date when a place was determined to be invalid (was never a "going concern").*

edtf:superseded:

> *Indicates the date when a record was superseded by another record.*

The why

The Extended Date/Time Format is a useful method to categorize "soft dates" for Who's On First, one example being the record for the country of Croatia as it relates to Yugoslavia. At the start of 1990, Croatia was a socialist republic within Yugoslavia. That same year, the government was dissolved in favor of a multi-party democracy and, soon after, an independence referendum was held. Independence from Yugoslavia was favored and within a month, Croatia declared independence.

This leaves us with a few questions. *Do we use the date of the independence vote as the inception date for Yugoslavia? Do we use the date of declared independence as the inception date? Is Croatia actually a separate "place" from Yugoslavia?* The answer to these questions is, well… yes, yes, and yes.

We can use *both* the date of the independence vote and the date of declared independence for Croatia's inception date using the Extended Date/Time Format. We know Croatia succeeded Yugoslavia, but the debate lies at *when* this took place. Using the date/string below allows us to be flexible, but still catalogue the “place” of Croatia:

```
"edtf:inception":"1991–05~/1991–06~"
```

That flexibility let’s Who’s On First to note the times around both dates related to Croatia’s independence.

Let’s take a stroll through history

1918-12~ to 1945-11-29

In December 1918, the **Kingdom of Yugoslavia (https://whosonfirst.mapzen.com/spelunker/id/1108955789)** came into existence after the dissolution of the Kingdom of Serbia and the Austro-Hungarian Empire. Also known as the “Kingdom of Serbs, Croats, and Slovenes” or simply “Yugoslavia”, the country was a combination of six republics: Bosnia & Herzegovina, Croatia, Macedonia, Montenegro, Serbia (including Kosovo), and Slovenia.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Click to view in full screen 🚀 (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/whosonfirst-edtf/yaml/yugo_1.yaml#6/44.292/-341.644)

1945-11-29 to 1991-06~

Following an **invasion and control by Axis powers (https://en.wikipedia.org/wiki/World_War_II_in_Yugoslavia)** between April 6, 1941 (1941-04-06) and November 29, 1945 (1945-11-29), the Kingdom of Yugoslavia was renamed the Federal People's Republic of Yugoslavia, then renamed the **Socialist Federal Republic of Yugoslavia (https://whosonfirst.mapzen.com/spelunker/id/1126113567)**.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Click to view in full screen ➶ (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/whosonfirst-edtf/yaml/yugo_2.yaml#6/44.292/-341.644)

1991-06~ to 1991-09~

Around June 1991, the former socialist republics of **Slovenia (https://whosonfirst.mapzen.com/spelunker/id/85633779)** and **Croatia (https://whosonfirst.mapzen.com/spelunker/id/85633229)** broke from the **Socialist Federal Republic of Yugoslavia (https://whosonfirst.mapzen.com/spelunker/id/1108955787)** to **form countries of their own (https://en.wikipedia.org/wiki/Breakup_of_Yugoslavia#Independence_of_Slovenia_and_Croatia)**.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Click to view in full screen ➶ (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/whosonfirst-edtf/yaml/yugo_3.yaml#6/44.292/-341.644)

1991-09~ to 1992-03~

Around September 1991, the former socialist republic of **Macedonia (https://whosonfirst.mapzen.com/spelunker/id/85632373)** broke from the **Socialist Federal Republic of Yugoslavia (https://whosonfirst.mapzen.com/spelunker/id/1108955785)**) to **form a country of its own (https://en.wikipedia.org/wiki/Breakup_of_Yugoslavia#Independence_of_the_Republic_of_Macedonia_and_Bosnia_and_Herzegovina)**.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Click to view in full screen ➶ (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/whosonfirst-edtf/yaml/yugo_4.yaml#6/44.292/-341.644)

1992–03~ to 2006

Around March 1992, the former socialist republic of **Bosnia & Herzegovina (https://whosonfirst.mapzen.com/spelunker/id/85632609)** broke from the **Socialist Federal Republic of Yugoslavia (https://whosonfirst.mapzen.com/spelunker/id/1108955783)** to **form a country of its own (https://en.wikipedia.org/wiki/Breakup_of_Yugoslavia#Independence_of_the_Republic_of_Macedonia_and_Bosnia_and_Herzegovina)**.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Click to view in full screen ➶ (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/whosonfirst-edtf/yaml/yugo_5.yaml#6/44.292/-341.644)

Present Day

The Socialist Federal Republic of Yugoslavia **dissolved in 2006 (https://en.wikipedia.org/wiki/Breakup_of_Yugoslavia#Aftermath_in_Serbia_and_Montenegro)** and, soon after, **Montenegro (https://whosonfirst.mapzen.com/spelunker/id/85632667)** joined Slovenia, Croatia, and Bosnia and Herzegovina in being internationally recognized as independent nations. **Kosovo (https://whosonfirst.mapzen.com/spelunker/id/85633259)** followed suit in 2008, with **Serbia (https://whosonfirst.mapzen.com/spelunker/id/85633755)** remaining as the successor state to Socialist Federal Republic of Yugoslavia.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
[Learn more.]

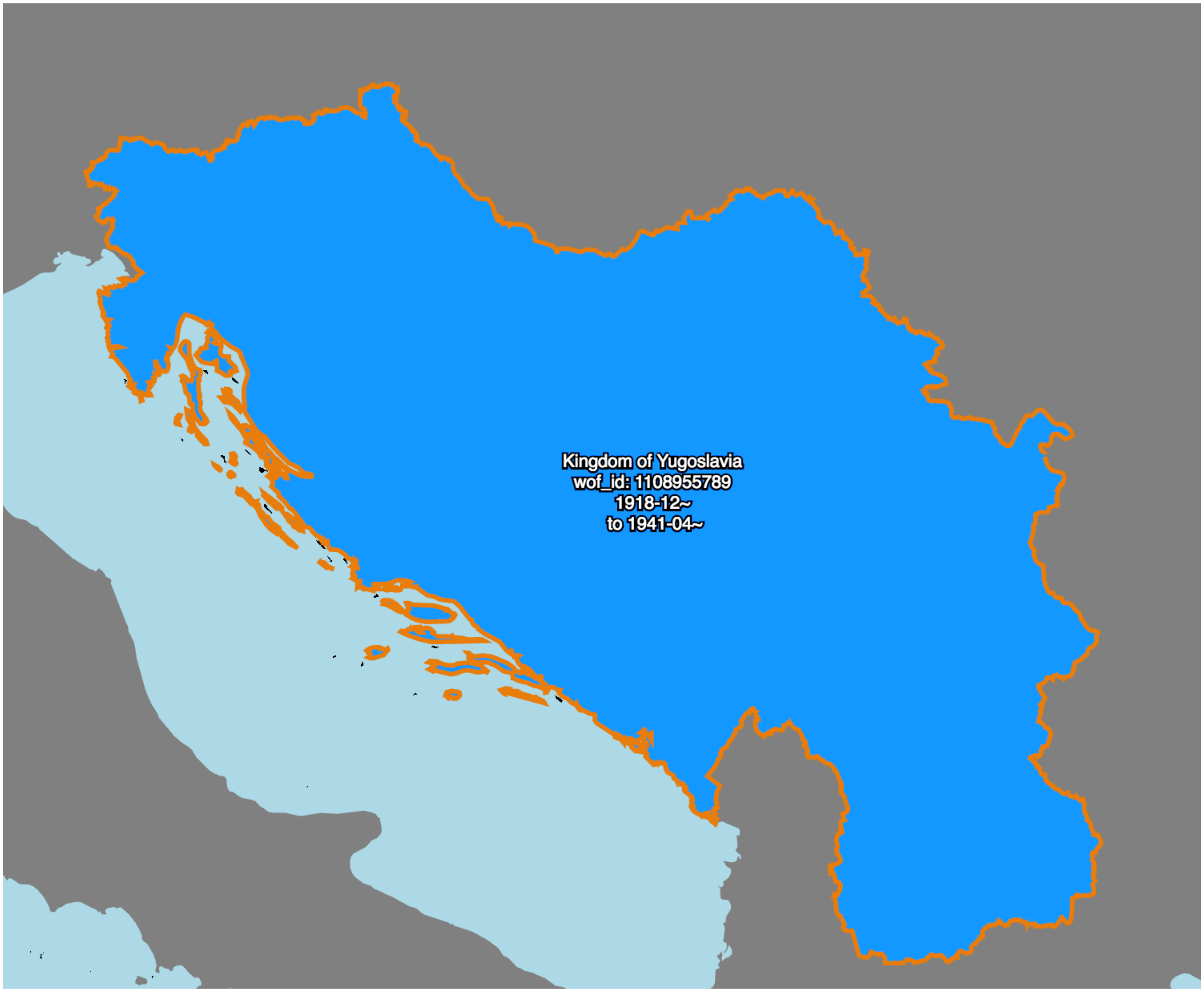
Click to view in full screen ➶ (https://tangrams.github.io/tangram-frame/?url=https://s3.amazonaws.com/mapzen-assets/images/whosonfirst-edtf/yaml/yugo_dissolved.yaml#6/44.292/-341.644)

You'll notice `supersedes` and `superseded_by` entries in WOF records. These are pointers to other WOF records: a current record can supersede the record of a place that is no longer a going concern, and a record not longer a going concern can be superseded by a current record. Looking at Yugoslavia, this is effectively a linked list:

```
1108955789 superseded by 1126113567
1126113567 superseded by 1108955787, 85633779, 85633229
1108955787 superseded by 1108955785, 85632373
1108955785 superseded by 1108955783, 85632609
1108955783 superseded by 85633755, 85632667, 85633259*
```

** Note: Kosovo was in a disputed state between 2006~ and 2008~*

Here is the dissolution of Yugoslavia in the form of a GIF, along with the WOF IDs and EDTF data:



You too can travel through time, space and politics by **superseding layers in Tangram Play (https://mapzen.com/tangram/view/?api=22/770#6.529/44.129/-341.711)**.

How does this relate to other records?

Great question!

Constituency records

We've recently imported constituency data for the 115th U.S. Congress (**take a look! (https://github.com/whosonfirst-data/whosonfirst-data-constituency-us)**) and have the scaffolding in place to import additional constituency data from previous sessions of Congress, as well as constituency data for other countries. While each Congress has a “hard” end date, the `edtf:*` properties will still be used signify dates in historical constituency records.

Manifest Destiny project

The **Manifest Destiny project (http://michaelporath.com/projects/manifest-destiny/)** is another interesting use case for the `edtf:*` properties. This project has geometries of unclaimed areas, claims, states, and territories in the United States from March 1789 to the present; this is a proposed import for Who’s On First and an interesting test case for more superseding and date classification work.

Existing records

We’ve done something similar with a smaller batch of records in San Francisco. Follow the `supersedes` links for two venues in San Francisco: **Rock Bar (https://whosonfirst.mapzen.com/spelunker/id/890535433/#14/37.7458/-122.4211)** and **Francisca’s (neé The Palace) (https://whosonfirst.mapzen.com/spelunker/id/1126130035/#13/37.7604/-122.4143)**.

Caveats…

If it isn’t obvious by now, tackling these types of issues can be challenging. We’re doing our best to catalogue historical places, but that comes with caveats. Using dates and superseding properties is helpful in creating a timeline of a “place”, but it is difficult to manage every edge case with the Extended Date/Time Format and other Who’s On First properties. Yugoslavia was an area with relative agreement when creating records and adding dates, but other places in long-term conflict will not be as easy to catalogue.

We welcome anyone to suggest improvements or methods to help us navigate through this historical work. If you have suggestions, questions, ideas, or comments, please **let use know (mailto:hello@mapzen.com)**!

Photo Credit: ***Iraia Martínez, Pixabay. (CC BY 2.0) (https://www.flickr.com/photos/aiari/3766024455/in/photolist-6JMRS8-6DruhW-dMpjq-qEC1wW-8jqbkg-3eKq11-9vJDWb-RYEBSv-erDaj-9UQbk-5Xx3no-5MLX4y-7SinmX-q9yT2J-hZMUDT-dEDz8e-VCw3RH-8o2aQe-d9txSf-Sb3iEj-8E1s88-SNrkgW-vESZ2-Uz71gq-Dn2miV-5YW8LF-TSNui7-B4uAyk-dbtWi5-71c6CD-KZsfb-oiYot-dAnKbo-a7Et8u-Av2fPK-bXANVG-9BGheC-bfxnzi-Uz77WE-o7oMjJ-DxpmsQ-7DPAkZ-Uz74ej-8XXBFV-dxVxBL-LqUFzX-Uvvtum-Uz79oY-oiJ5aM-dm4Qmn)***

· 29 June 2017 ·

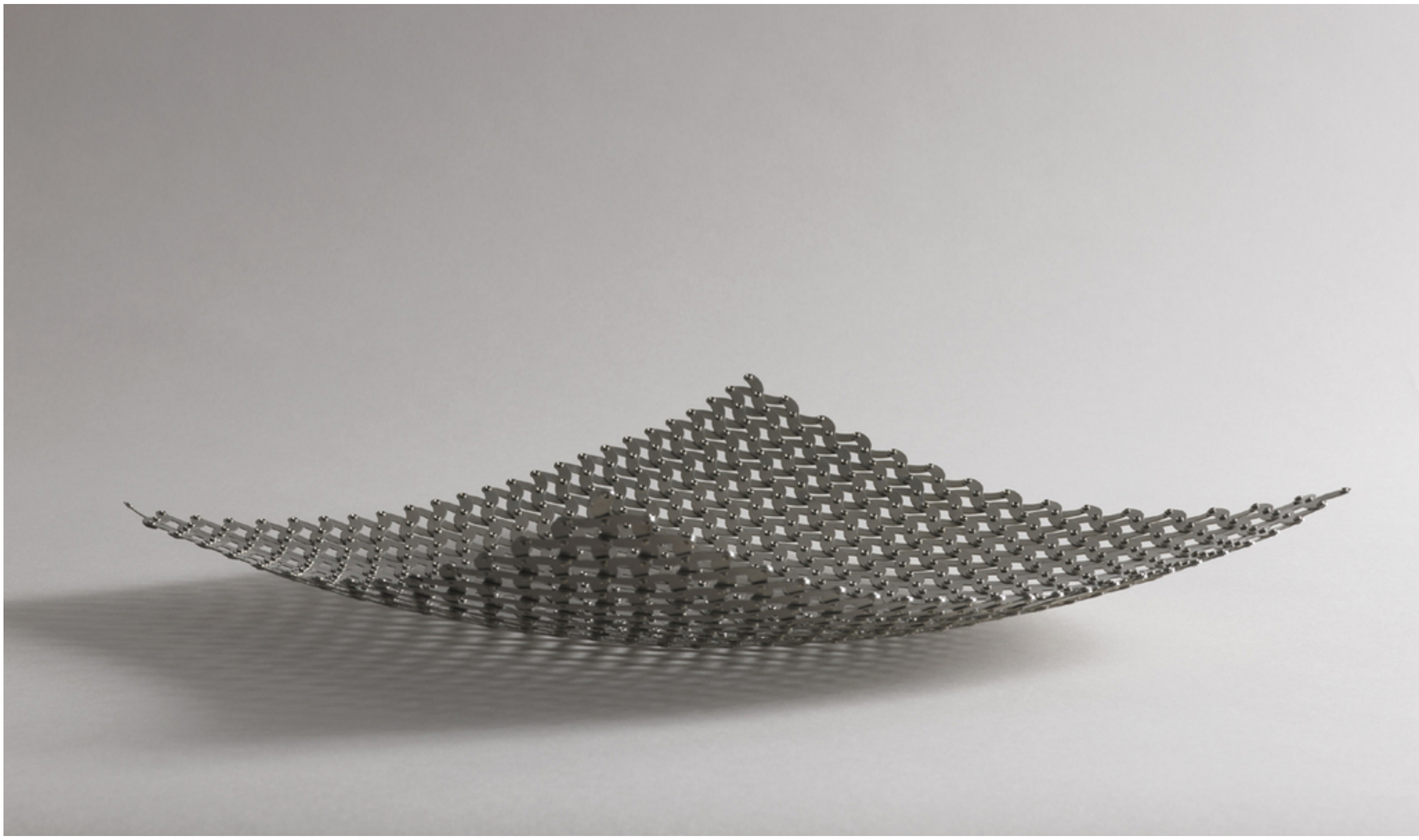
Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.

© 2017 Mapzen

Mapzen Flex expands your travel horizons!

flex (/tag/flex) mobility (/tag/mobility)



Last month we introduced **Mapzen Flex (https://mapzen.com/pricing)**, giving you fixed, transparent prices across our products. Instead of paying for a bundle you may not need, or getting locked into a package or tier, you just pay a modest amount per thousand requests for what you use above the free monthly limit.

In August, we're adding four services to Mapzen Flex:

- **Time-distance matrix (https://mapzen.com/blog/matrix/)**
- **Isochrone (https://mapzen.com/blog/introducing-isochrone-service/)**
- **Optimized route (https://mapzen.com/blog/optimized-route/)**
- **Elevation (https://mapzen.com/blog/moving-on-up/)**

Just like before, each service will have a free level which is enough to experiment and get a feel for the service. If you don't go above that, not much will change. The difference? Now you can pay to exceed the free request cap. Use as little or as as much as you need each month, and set spending limits to keep your bill within your budget.

You can check out the new pricing on the **Mapzen Flex (https://mapzen.com/pricing)** page. We'll be rolling out the new rate limits for these services on **August 1st**. Drop us a line at **hello@mapzen.com (mailto:hello@mapzen.com)** if you have any questions!

· 30 June 2017 ·

Mapzen

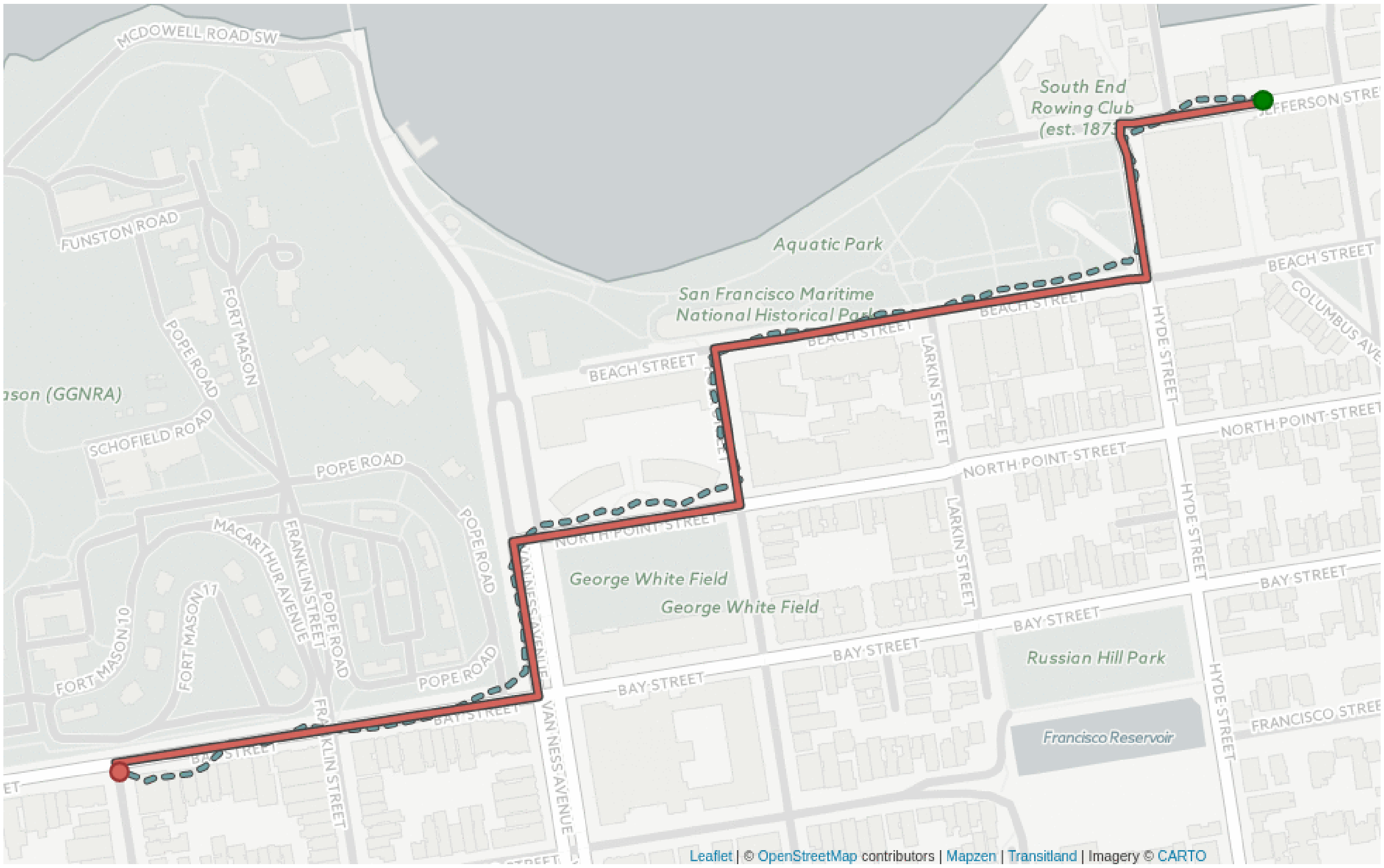
Opening maps with open source and open data.

© 2017 Mapzen

Map Matching is ready to Flex

flex (/tag/flex) routing (/tag/routing) mobility (/tag/mobility)

We're excited to release Mapzen Map Matching, the latest addition to the Mapzen Mobility services. Our Map Matching service takes in a trajectory of latitude/longitude coordinates and returns back a route line that is snapped to the OpenStreetMap road network and enriched with attributes, like speed limits and freeway exit signs.


MCDOWELL ROAD SW
FUNSTON ROAD
South End
Rowing Club
(est. 1873)
JEFFERSON STREET
Aquatic Park
San Francisco Maritime
National Historical Park
BEACH STREET
BEACH STREET
BEACH STREET
COLUMBUS AVE
FORT MASON
POPE ROAD
ason (GGNRA)
SCHOFIELD ROAD
POPE ROAD
LARKIN STREET
HYDE STREET
NORTH POINT STREET
NORTH POINT STREET
NORTH POINT STREET
MACARTHUR AVENUE
FRANKLIN STREET
POPE ROAD
POPE ROAD
George White Field
George White Field
FORT MASON 17
FORT MASON 10
FORT MASON
VAN NESS AVENUE
BAY STREET
BAY STREET
BAY STREET
BAY STREET
BAY STREET
Russian Hill Park
Francisco Reservoir
FRANCISCO STREET
FRANKLIN STREET
VAN NESS AVENUE

Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery © CARTO

Mapzen Map Matching can be used to power many applications, from GPS fitness trackers to heads-up navigation displays. Starting today, it's available through a **generous free tier to all developers (https://mapzen.com/dashboard)** and with **the simple pricing of Mapzen Flex for heavier users (/pricing)**.

In this blog post, we'll provide some more examples of what Mapzen Map Matching can do. We'll explain its two specific functions: `trace_attributes` and `trace_route` . And we'll provide all the information you need to start using Mapzen Map Matching yourself in the Mapzen Mobility

Explorer client and directly from the Map Matching API!

Trace Attributes

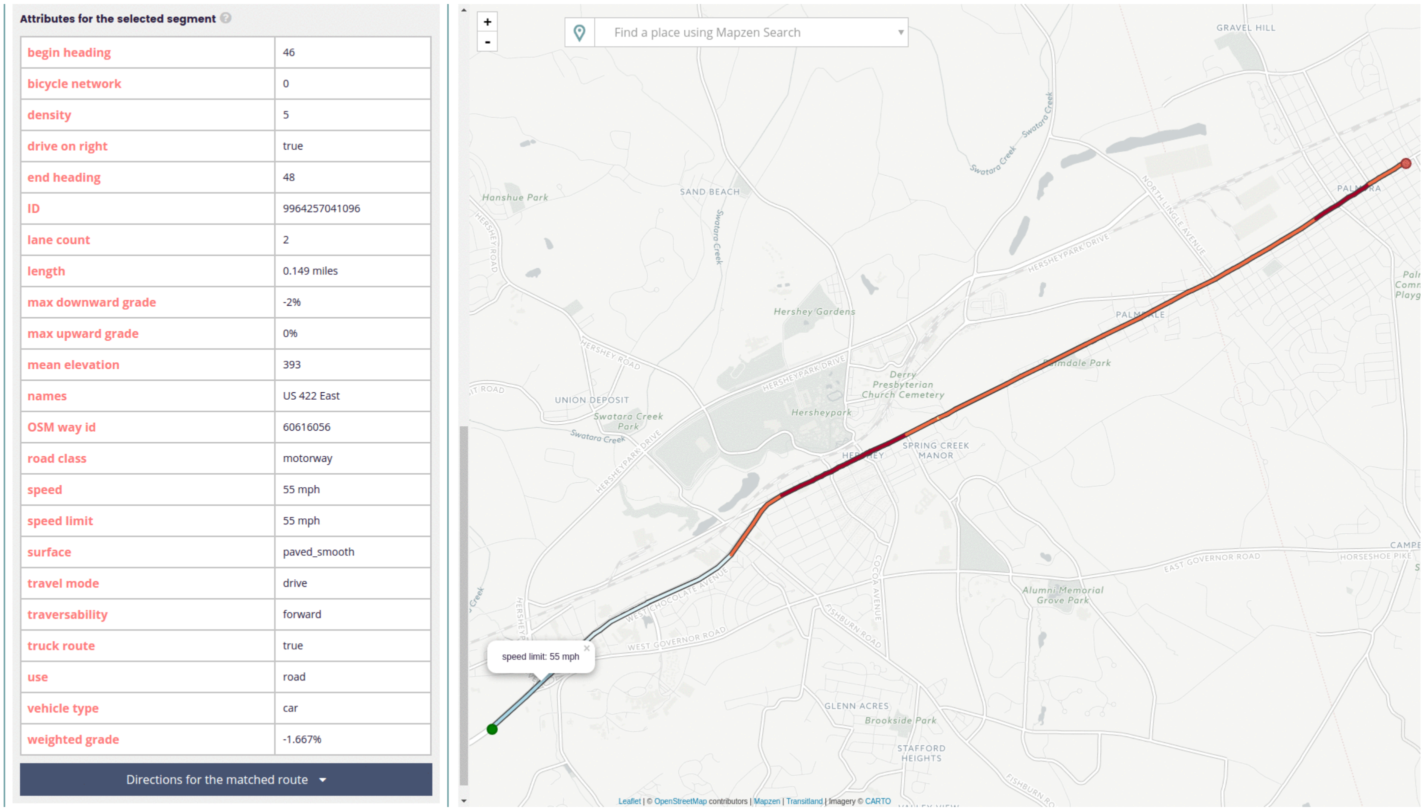
Let us state a few scenarios to better describe our `trace_attributes` functionality…

Are you on a road trip and are wondering about what lies ahead? Are you curious about the speed limit or tolls along your trip? Maybe you are visiting your grandmother, who uses a wheelchair, and would like to take her out for a stroll. Wouldn't it be useful to know the upward or downward slope along your walking route?

A few common inquiries on attributes along a route:

- Types of bike lanes
- Signage information for exits (exit name, exit number, exit branch)
- Speed limits
- Maximum upward or download slope along the bike or walking route
- Any tolls, tunnels, or bridges along the route
- Any unpaved sections of roads

The `trace_attributes` action can give you these sorts of localized descriptions of your current route segment or a portion of what is ahead on your route. Here's a **full list (https://mapzen.com/documentation/mobility/map-matching/api-reference/#attribute-filters-trace_attributes-only)** of the attributes that can be requested. In addition, `trace_attributes` queries return the matched shape path, which you can use to draw on a map:



Attributes for the selected segment

begin heading	46
bicycle network	0
density	5
drive on right	true
end heading	48
ID	9964257041096
lane count	2
length	0.149 miles
max downward grade	-2%
max upward grade	0%
mean elevation	393
names	US 422 East
OSM way id	60616056
road class	motorway
speed	55 mph
speed limit	55 mph
surface	paved_smooth
travel mode	drive
traversability	forward
truck route	true
use	road
vehicle type	car
weighted grade	-1.667%

Directions for the matched route
+
-
Find a place using Mapzen Search
speed limit: 55 mph
GRAVEL HILL
SAND BEACH
Hanshue Park
Hershey Gardens
Hersheypark
UNION DEPOSIT
Swatara Creek Park
Derry Presbyterian Church Cemetery
SPRING CREEK MANOR
Alumni Memorial Grove Park
GLENN ACRES
Brookside Park
STAFFORD HEIGHTS
EAST GOVERNOR ROAD
WEST GOVERNOR ROAD
Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery © CARTO

(Open full screen ➶ (https://s3.amazonaws.com/mapzen-assets/images/map-matching/hershey-route-speeds.gif))

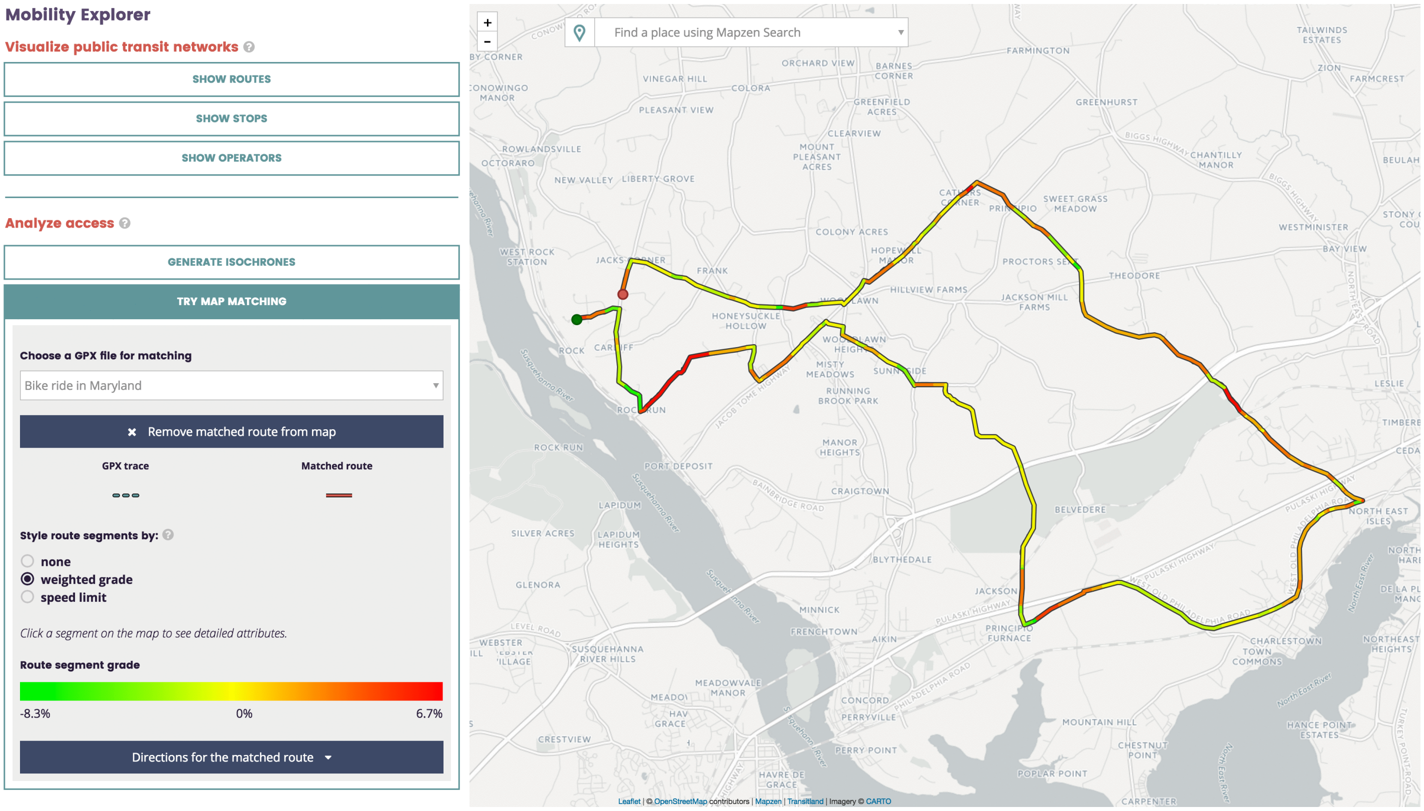
Trace Route

Ok, so now that you get the gist of `trace_attributes`, let us move on to our second really awesome `trace_route` function.

It’s going to be a beautiful weekend, perfect for an outdoor adventure. It could consist of hiking, biking or just a nice drive in the country to unwind from the hectic work week. What if your amazing adventure could benefit someone else? How many times have you found a great path or route that you would love to record to use again or even share with others?

The `trace_route` action allows you to do just that. Our service can take the trajectory points and match them to a Valhalla route path, along with providing you a step-by-step navigation for your map-matched path (just like a route planned using **Mapzen Turn-by-Turn (/products/turn-by-turn/)**).

What’s even better is that you can now visualize these cool new features within our Mobility Explorer! The Explorer will allow you to **upload your favorite GPX trace (https://mapzen.com/documentation/mobility/explorer/map-matching/#match-a-gpx-file)** to test out our Map Matching service. (GPX is a common file format that many GPS loggers, running watches, and other consumer electronics can output.) The Mobility Explorer will first perform a `trace_attributes` on your trace and can provide you with a clickable route highlight based on speed or grade to display various attributes along your path. If there are no errors in the matched data, the Explorer will also provide you with navigational output for the matched route. Please **try them out (https://mapzen.com/mobility/explorer/#/map-matching)** for yourself!



Choose a GPX file for matching: Bike ride in Maryland

Style route segments by:

○ none
● weighted grade
○ speed limit

Click a segment on the map to see detailed attributes.

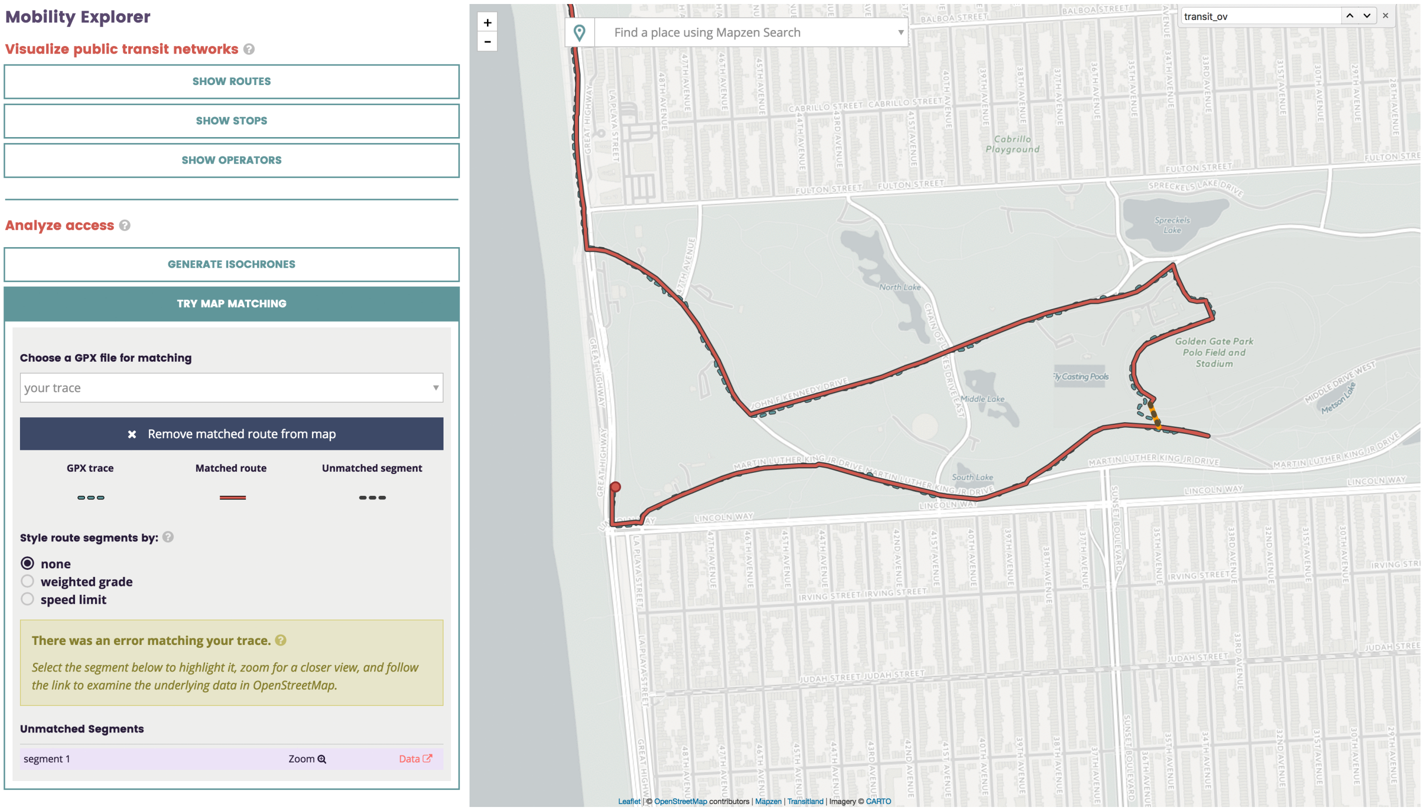
Mobility Explorer
Visualize public transit networks
SHOW ROUTES
SHOW STOPS
SHOW OPERATORS
Analyze access
GENERATE ISOCHRONES
TRY MAP MATCHING
Remove matched route from map
GPX trace
Matched route
Route segment grade
-8.3%
0%
6.7%
Directions for the matched route
Find a place using Mapzen Search
FARMINGTON
ORCHARD VIEW
BARNES CORNER
VINEGAR HILL
COLORA
GREENHURST
GREENFIELD ACRES
PLEASANT VIEW
CLEARVIEW
ROWLANDSVILLE
OCTORARO
MOUNT PLEASANT ACRES
CHANTILLY MANOR
NEW VALLEY
LIBERTY GROVE
SWEET GRASS MEADOW
COLONY ACRES
WESTMINSTER
WEST ROCK STATION
FRANK
PROCTORS SE
THEODORE
HILLVIEW FARMS
HONEYSUCKLE HOLLOW
JACKSON MILL FARMS
MISTY MEADOWS
RUNNING BROOK PARK
ROCK RUN
MANOR HEIGHTS
PORT DEPOSIT
CRAIGTOWN
BELVEDERE
LAPIDUM
LAPIDUM HEIGHTS
SILVER ACRES
BLYTHEDALE
NORTH EAST ISLES
GLENORA
JACKSON
MINNICK
FRENCHTOWN
AIKIN
PRINCIPIO FURNACE
CHARLESTOWN TOWN COMMONS
WEBSTER
SUSQUEHANNA RIVER HILLS
MEADOWVALE MANOR
CONCORD
PERRYVILLE
MOUNTAIN HILL
CHESTNUT POINT
HANCE POINT ESTATES
CRESTVIEW
PERRY POINT
HAVRE DE GRACE
POPLAR POINT
CARPENTER
PULASKI HIGHWAY
Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery © CARTO

(Open full screen ➶ (https://s3.amazonaws.com/mapzen-assets/images/map-matching/maryland-bike-route-match-gradient.gif))

Your mileage may vary

Map matching is a probabilistic process. Depending upon the accuracy and precision of the data coming in, there is always some chance of error in the match data that our service will send back. When errors do occur, our Map Matching service will provide as much information as possible to help you understand the source of the error and how you may be able to fix it. The results may be incomplete because there are no lines to match to the underlying OpenStreetMap source data. When the match produces an error or a discontinuity, you can style the segments by attribute but you will not be able to get turn-by-turn directions for your route

Here is an example from the Mobility Explorer of an unsuccessful match. The Mobility Explorer will output a list of unmatched segments, along with links to zoom to them on the map and see the areas in OpenStreetMap:


Mobility Explorer
Visualize public transit networks
SHOW ROUTES
SHOW STOPS
SHOW OPERATORS
Analyze access
GENERATE ISOCHRONES
TRY MAP MATCHING
Remove matched route from map
GPX trace
Matched route
Unmatched segment
Find a place using Mapzen Search
transit_ov
Golden Gate Park Polo Field and Stadium
Spreckels Lake
North Lake
Middle Lake
South Lake
Cabrillo Playground
Fly Casting Pools
Leaflet | © OpenStreetMap contributors | Mapzen | Transitland | Imagery © CARTO

Choose a GPX file for matching

your trace

Style route segments by:

● none
○ weighted grade
○ speed limit

There was an error matching your trace.

Select the segment below to highlight it, zoom for a closer view, and follow the link to examine the underlying data in OpenStreetMap.

Unmatched Segments

segment 1 Zoom Data

(Open full screen ➶ (https://s3.amazonaws.com/mapzen-assets/images/map-matching/golden-gate-park-route-discontinuity.gif))

Suggestions for improving your results

Follow these guidelines to improve the Map Matching results.

- You should have good GPS accuracy, with a clear view of the sky and as many satellites as possible. Of course, this can be difficult in dense urban areas.
- Make sure the trace point density is within the approximate range of one per second and one per 10 seconds. The greater the range between them, the more chance of inaccurate results.
- Have each trace represent one continuous path.
- Verify that there is a corresponding match with the OpenStreetMap network. If there are missing roadways/paths or improper connections, you can always use your local knowledge to fix and improve OpenStreetMap!

Did Someone Say Open Traffic?

The Mapzen Map Matching service has many more applications. We're also using the service to turn raw GPS probe data from vehicles into traffic speed statistics. The Open Traffic platform collects these traffic speed statistics from around the globe, in a manner that protects the privacy of individuals and companies that contribute their anonymous location data.

Read more about Open Traffic (/blog/announcing-open-traffic/), a collaboration between the World Bank, Mapzen, and ride-sharing companies around the world.

A Note of Appreciation

Finally, we'd like to thank Mapillary and especially Tao Peng. Valhalla's core map-matching component (called Meili) has been a collaborative creation with them **since the start (/blog/mapillary/)**. Also, we'd like to give a tip of the hat to Paul Newson and John Krumm, creators of the **Hidden Markov Model approach to map-matching (https://github.com/valhalla/valhalla/blob/master/docs/meili/algorithms.md)** while at Microsoft Research in 2009—it's this approach that we've implemented in Meili. Just like all of Valhalla, Meili is open-source. We look forward to continuing to make improvements to its abilities and applications with our friends at Mapillary and any other contributors who want to join us **on GitHub (https://github.com/valhalla/valhalla)**!

We have a thorough explanation of our Map Matching Service that you can access by going to our **Map Matching documentation (https://mapzen.com/documentation/mobility/map-matching/api-reference)**. You can also view a live demonstration of our Map Matching Service by trying out our **Mobility Explorer (https://mapzen.com/mobility/explorer)**. We love feedback so send it **our way (mailto:hello@mapzen.com)**!

Map Matching is available immediately through **Mapzen Flex (https://mapzen.com/pricing)**, with a generous free tier and **simple and understandable prices (https://mapzen.com/pricing)** if you need to match and get attributes on even more routes.

· 10 July 2017 ·

Kristen DiLuca

Kristen is a software engineer specializing in our routing API services. She also enjoys dabbling in javascript for our open source routing test tools and always welcomes new challenges.

Meghan Hade

Meghan is a software engineer with a background in urban planning on Mapzen's Mobility team. She works in javascript and plays in calligraphy, block printing, and finding ways to stash n+1 bikes in a San Francisco apartment.

Duane Gearhart

Duane is a software engineer @mapzen specializing in quality route guidance and real-world route analysis.

© 2017 Mapzen

Refill – Levels of Detail

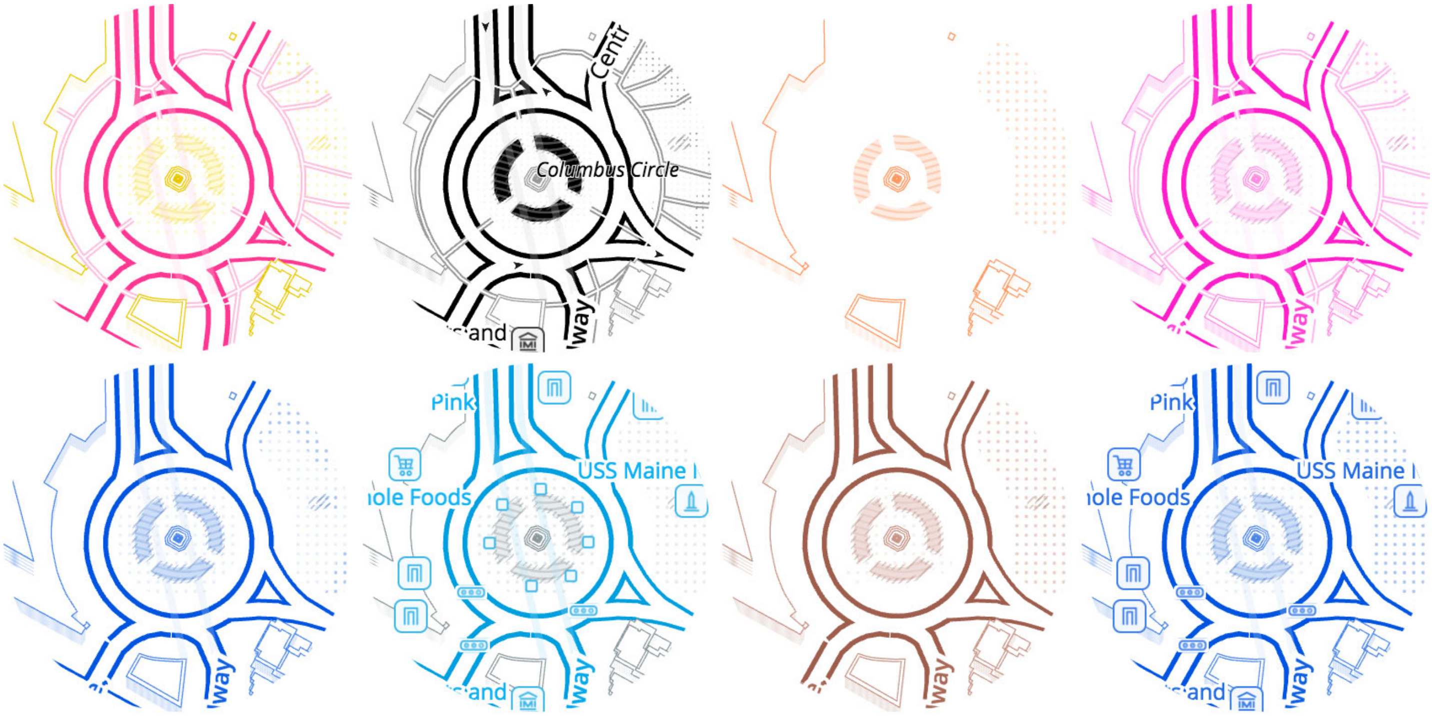
cartography (/tag/cartography) tangram (/tag/tangram) vector-tiles (/tag/vector-tiles)

Customize Refill

Last month we introduced Refill **themes** in **dark (https://mapzen.com/blog/refill-themes-part-2/)** and **light (https://mapzen.com/blog/refill-themes/) color** palettes. This month we're happy to give you more ways to customize Refill style with two more theme options: new levels of **detail** and additional **label** steps.

Sometimes *less* is more. Now when you remix Refill with data visualization overlays you can tone down the number of features shown in the basemap to let the overlay sing. Map still too busy? Cut down on the number of basemap labels.

*(More on using Mapzen basemaps in Tangram for data visualization is available in our **Lots of Dots (https://mapzen.com/blog/lots-of-dots/)**, **Map Sandwich (https://mapzen.com/blog/map-sandwich)**, and others in our **Make Your Own (https://mapzen.com/tag/make-your-own/)** blog post series.)*



Columbus Circle drawn in a variety of ways. ***View Map (https://mapzen.com/products/maps/refill/detail/?detail=10&label=10&color=pink-yellow#lat=40.76799&lng=286.01808&z=17)***

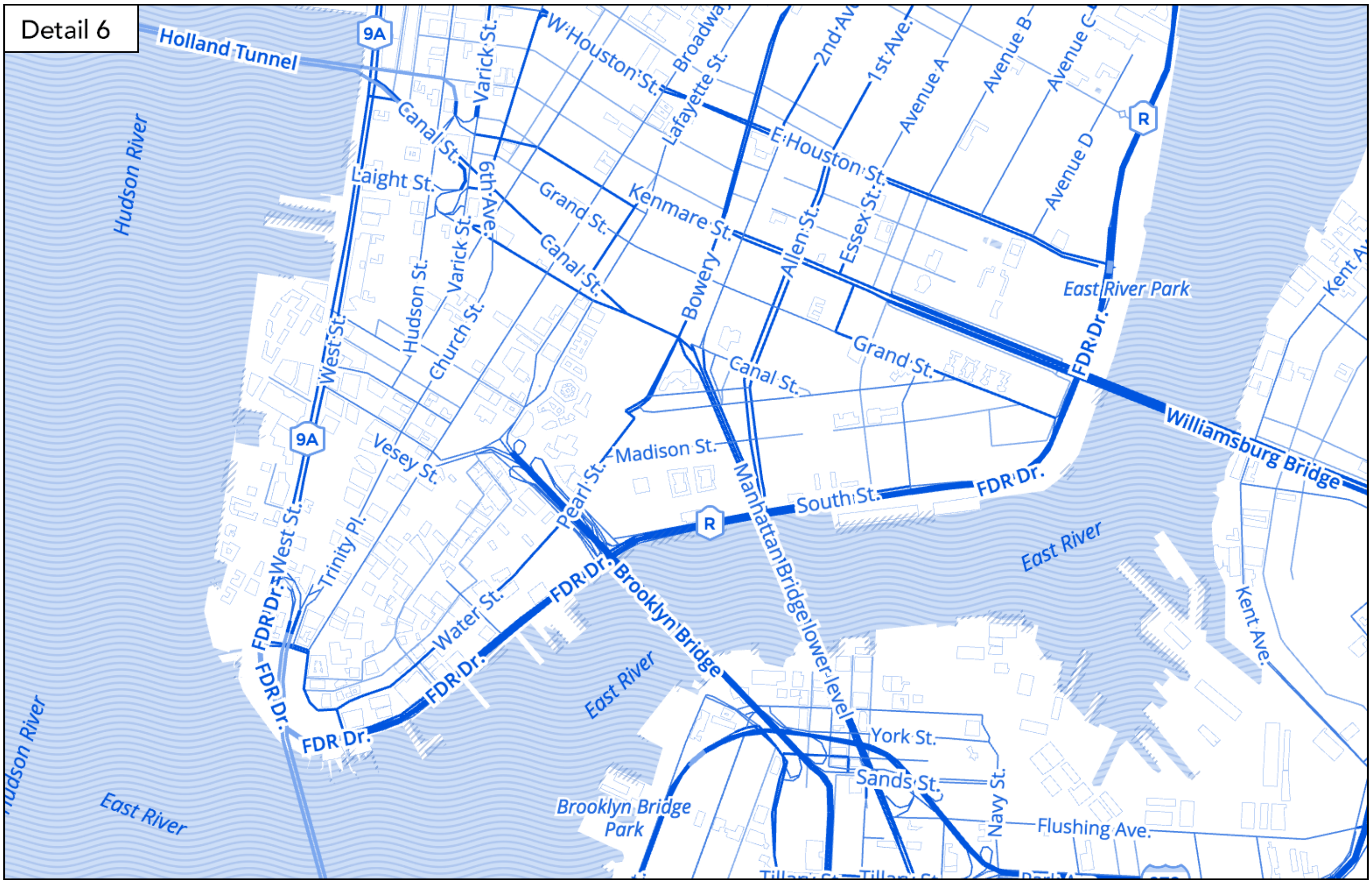
Levels of Detail

We’ve taken Refill labels **up to eleven (https://en.wikipedia.org/wiki/Up_to_eleven)** in this release with a “show me everything” option at level **11**. There are now **12 levels of detail** total ranging from 0 to 11. Higher detail levels show more map content, while the lower levels show less. The default level of detail in Refill remains the same at **detail-10**.

Because Mapzen **vector tiles (https://mapzen.com/documentation/vector-tiles/layers/)** include a `min_zoom` property on each feature, marking which zoom the feature started appearing in Mapzen tiles, we can adjust our client-side Tangram filters to say for *this* zoom, only show features that were visible if they were visible several zooms *earlier*.

```
roads:
    filter: function() { return feature.min_zoom <= ($zoom - 2) }
```

We also disable certain kinds of features at lower levels of detail by category.

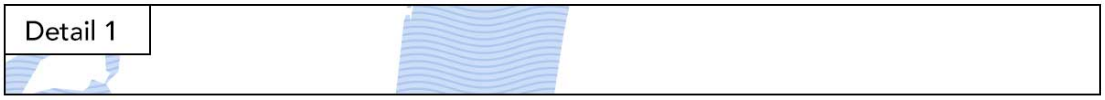

Detail 6
Holland Tunnel
9A
Hudson River
W Houston St.
Broadway
Lafayette St.
2nd Ave.
1st Ave.
Avenue A
Avenue B
Avenue C
Avenue D
E Houston St.
R
Canal St.
Varick St.
6th Ave.
Laight St.
Grand St.
Kenmare St.
Allen St.
Essex St.
East River Park
Hudson St.
Church St.
Bowery
West St.
Grand St.
FDR Dr.
Canal St.
Williamsburg Bridge
Vesey St.
Madison St.
Manhattan Bridge
South St.
FDR Dr.
Pearl St.
Trinity Pl.
West St.
East River
Kent Ave.
Water St.
FDR Dr.
Brooklyn Bridge
Manhattan Bridge lower level
East River
York St.
Sands St.
Navy St.
Flushing Ave.
Brooklyn Bridge Park
FDR Dr.
Hudson River
East River

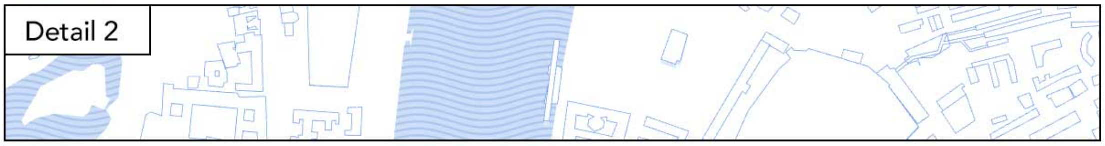
New York at zoom-14 — detail-0 to detail-11 — Blue Refill Theme. ***View Map*** *(https://mapzen.com/products/maps/refill/detail/?detail=11&label=10&color=blue#lat=40.7139&lng=-74.0019&z=14)*

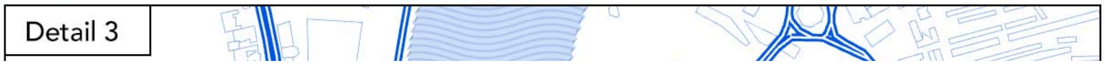
We start with *water* and *earth* at **detail-0**, with *inland water* added at **detail-1**. Then *buildings* and *ocean water boundaries* come into view at **detail-2**.

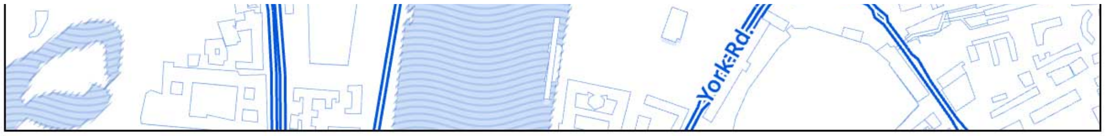
Roads begin at **detail-3**, starting with *highways*. *Other water boundaries* start to show at this level of detail as well. *Major roads* and *place labels* appear at **detail-4** and gradually build up as you go further up the levels.

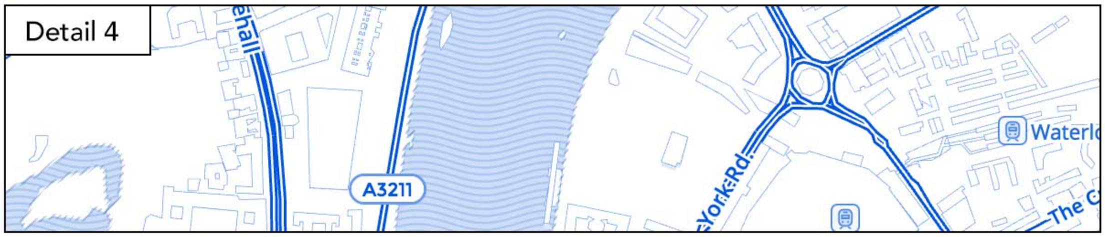
View the progression of details at Waterloo station and the River Thames in London at zoom level 15.

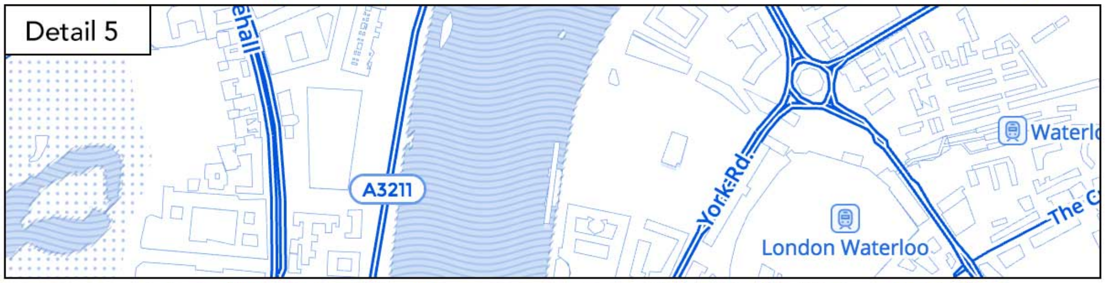


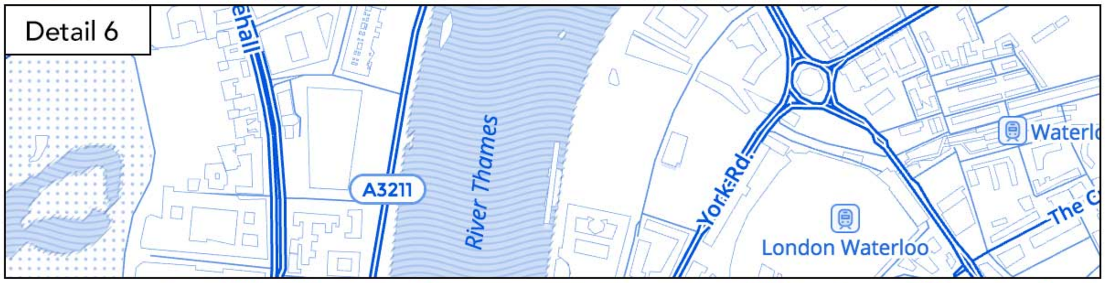


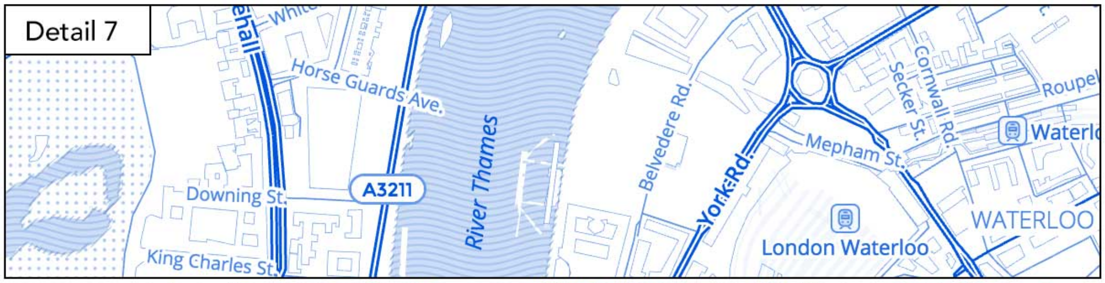


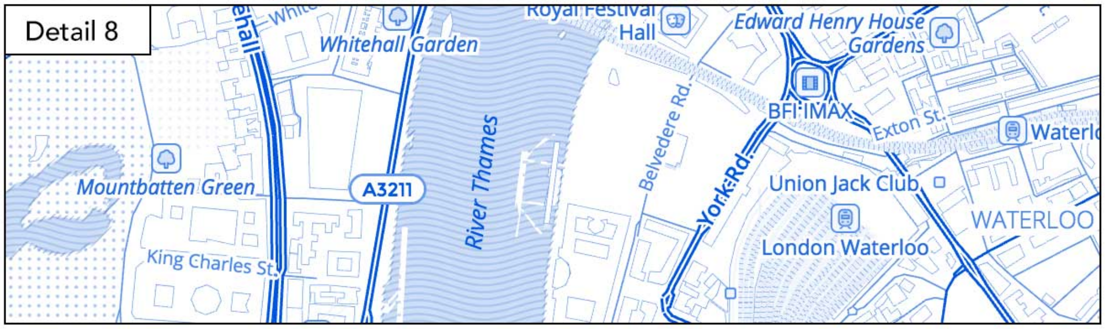


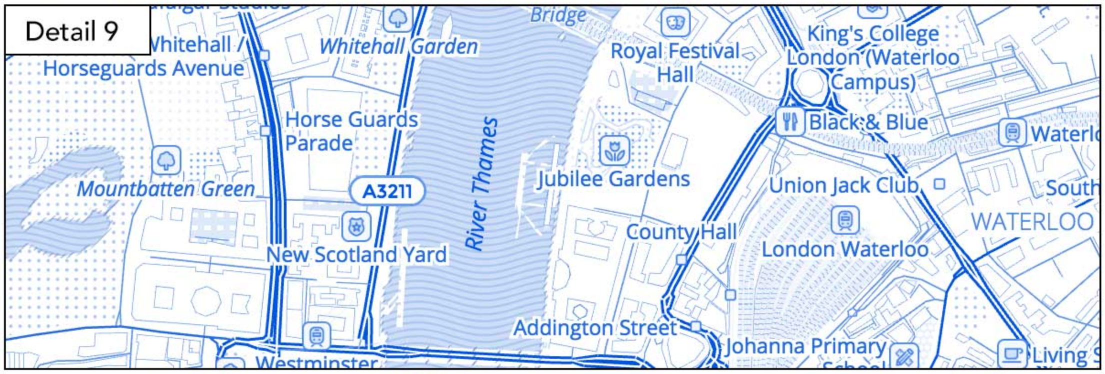

Detail 4
ehall
A3211
York Rd.
Waterl
The C


Detail 5
ehall
A3211
York Rd.
Waterl
London Waterloo
The C

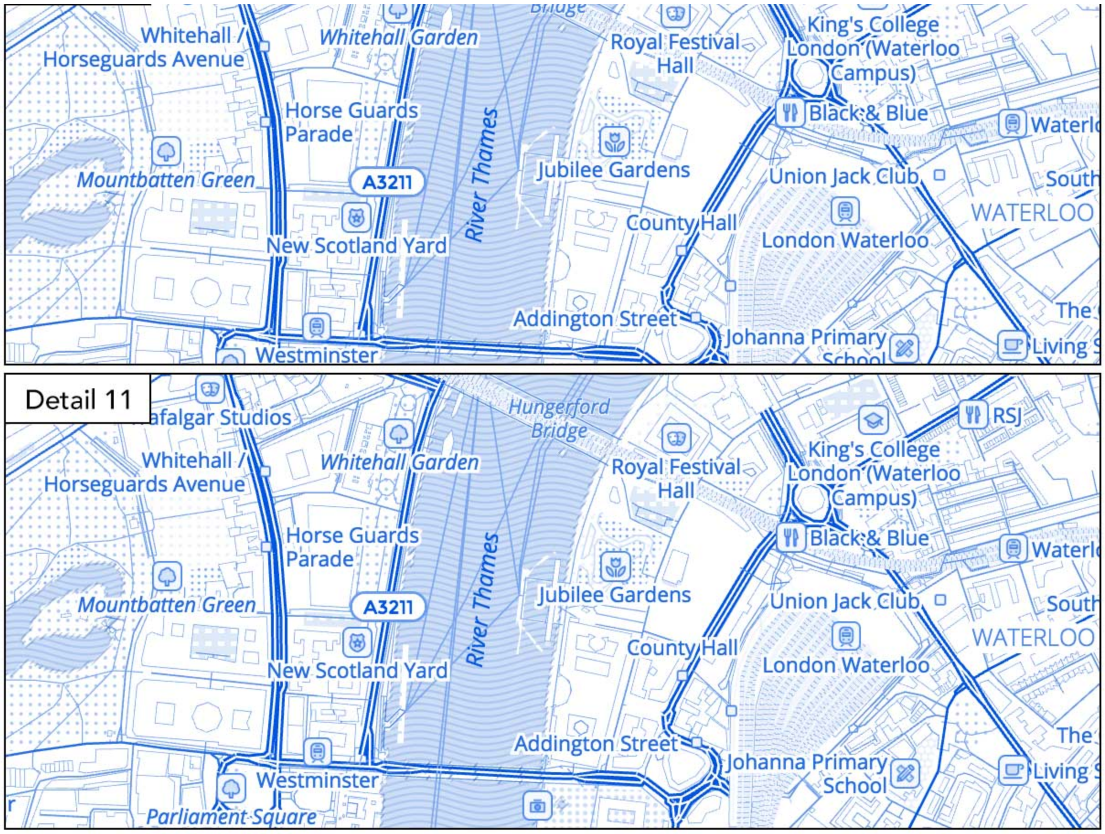

Detail 6
ehall
A3211
River Thames
York Rd.
Waterl
London Waterloo
The C


Detail 7
ehall
White
Horse Guards Ave.
Downing St.
King Charles St.
A3211
River Thames
Belvedere Rd.
York Rd.
Mepham St.
Cornwall Rd.
Secker St.
Roupel
Waterl
London Waterloo
WATERLOO


Detail 8
ehall
White
Whitehall Garden
Royal Festival Hall
Edward Henry House Gardens
BFI IMAX
Exton St.
Waterl
Mountbatten Green
King Charles St.
A3211
River Thames
Belvedere Rd.
York Rd.
Union Jack Club
London Waterloo
WATERLOO

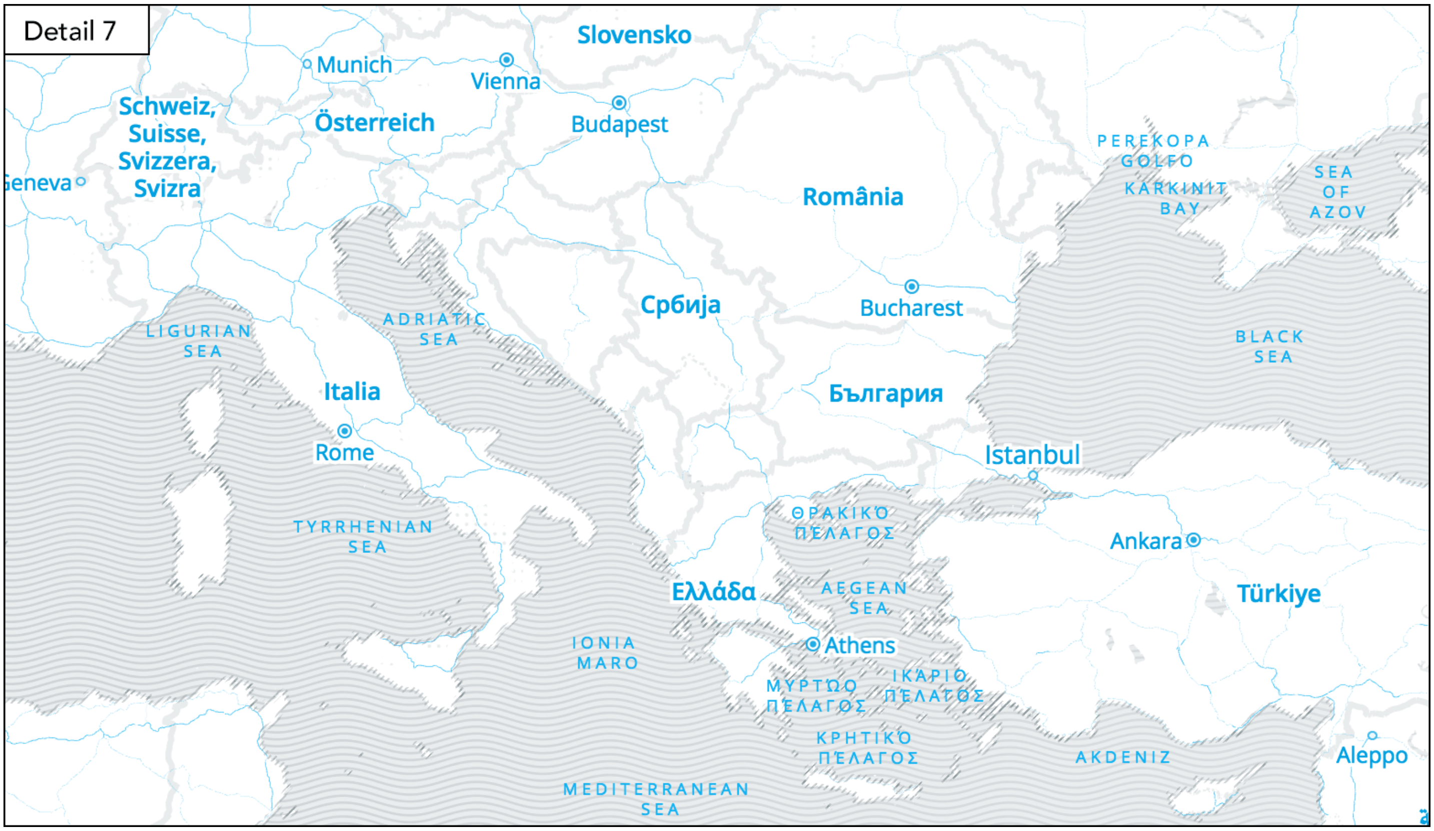

Detail 9
afalgar Studios
Whitehall / Horseguards Avenue
Whitehall Garden
Bridge
Royal Festival Hall
King's College London (Waterloo Campus)
Horse Guards Parade
Black & Blue
Waterl
Mountbatten Green
A3211
River Thames
Jubilee Gardens
Union Jack Club
South
WATERLOO
New Scotland Yard
County Hall
London Waterloo
Addington Street
Johanna Primary School
Living S
Westminster

Detail 10
afalgar Studios
Hungerford Bridge
RSJ


Detail 11
Whitehall / Horseguards Avenue
Whitehall Garden
Horse Guards Parade
Mountbatten Green
A3211
New Scotland Yard
River Thames
Westminster
Parliament Square
Hungerford Bridge
Royal Festival Hall
Jubilee Gardens
County Hall
Addington Street
King's College London (Waterloo Campus)
Black & Blue
Union Jack Club
London Waterloo
WATERLOO
Johanna Primary School
RSJ
Trafalgar Studios

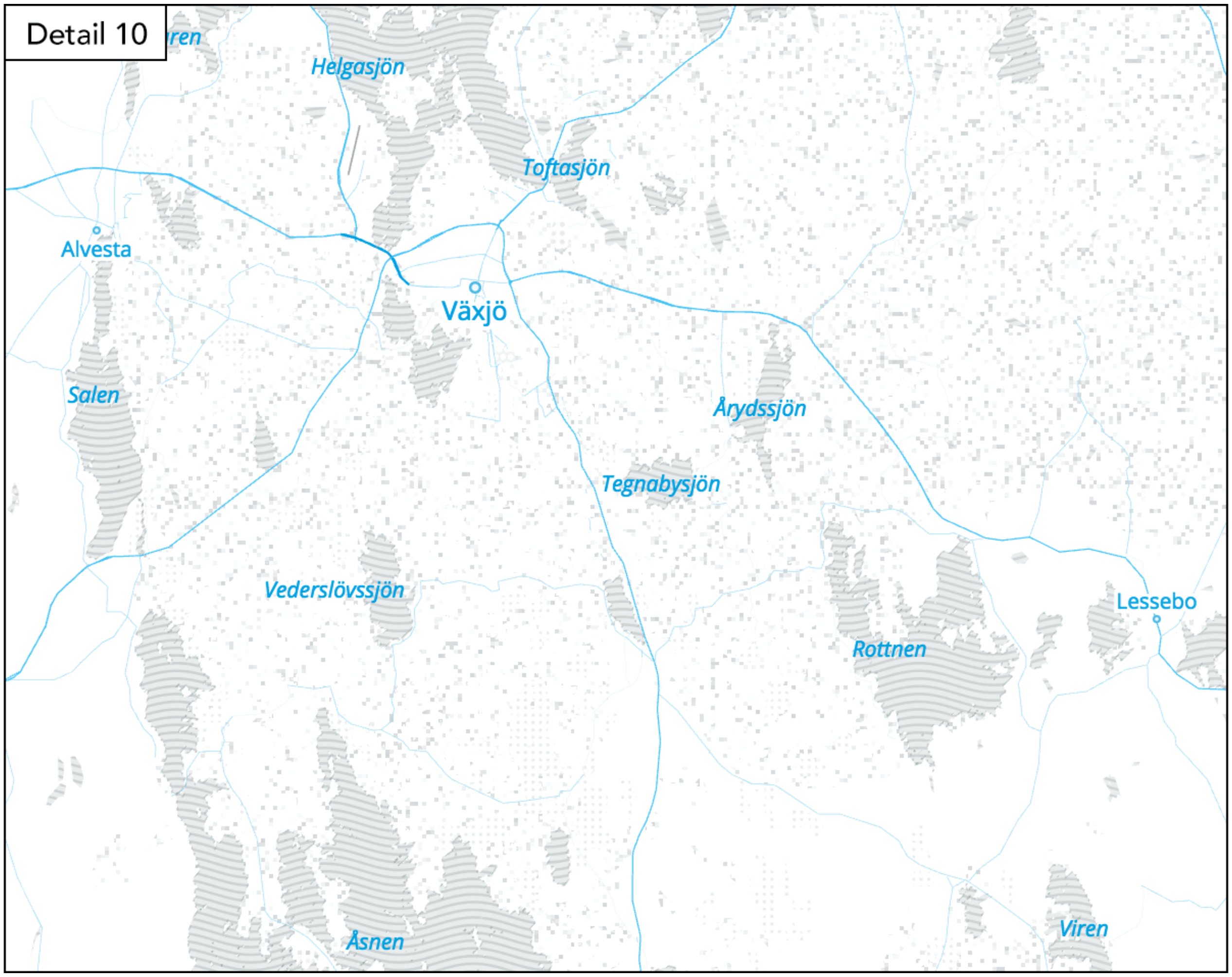
London at zoom-15 — detail-0 to detail-11 + label-11 — Blue Refill Theme. ***View Map (https://mapzen.com/products/maps/refill/detail/?detail=10&label=11&color=blue#lat=51.5039&lng=-0.1193&z=15)***

Here’s Eastern Europe and the Mediterranean at zoom level 5. *Country borders* begin at **detail-5** and then *country labels* follow at **detail-7**.


Detail 7
Slovensko
Munich
Vienna
Schweiz,
Suisse,
Svizzera,
Svizra
Österreich
Budapest
Geneva
România
PEREKOPA
GOLFO
KARKINIT
BAY
SEA
OF
AZOV
LIGURIAN
SEA
ADRIATIC
SEA
Србија
Bucharest
BLACK
SEA
Italia
България
Rome
Istanbul
ΘΡΑΚΙΚΟ
ΠΕΛΑΓΟΣ
TYRRHENIAN
SEA
Ankara
Ελλάδα
AEGEAN
SEA
Türkiye
IONIA
MARO
Athens
ΜΥΡΤΩΟ
ΠΕΛΑΓΟΣ
ΙΚΑΡΙΟ
ΠΕΛΑΓΟΣ
ΚΡΗΤΙΚΟ
ΠΕΛΑΓΟΣ
AKDENIZ
Aleppo
MEDITERRANEAN
SEA

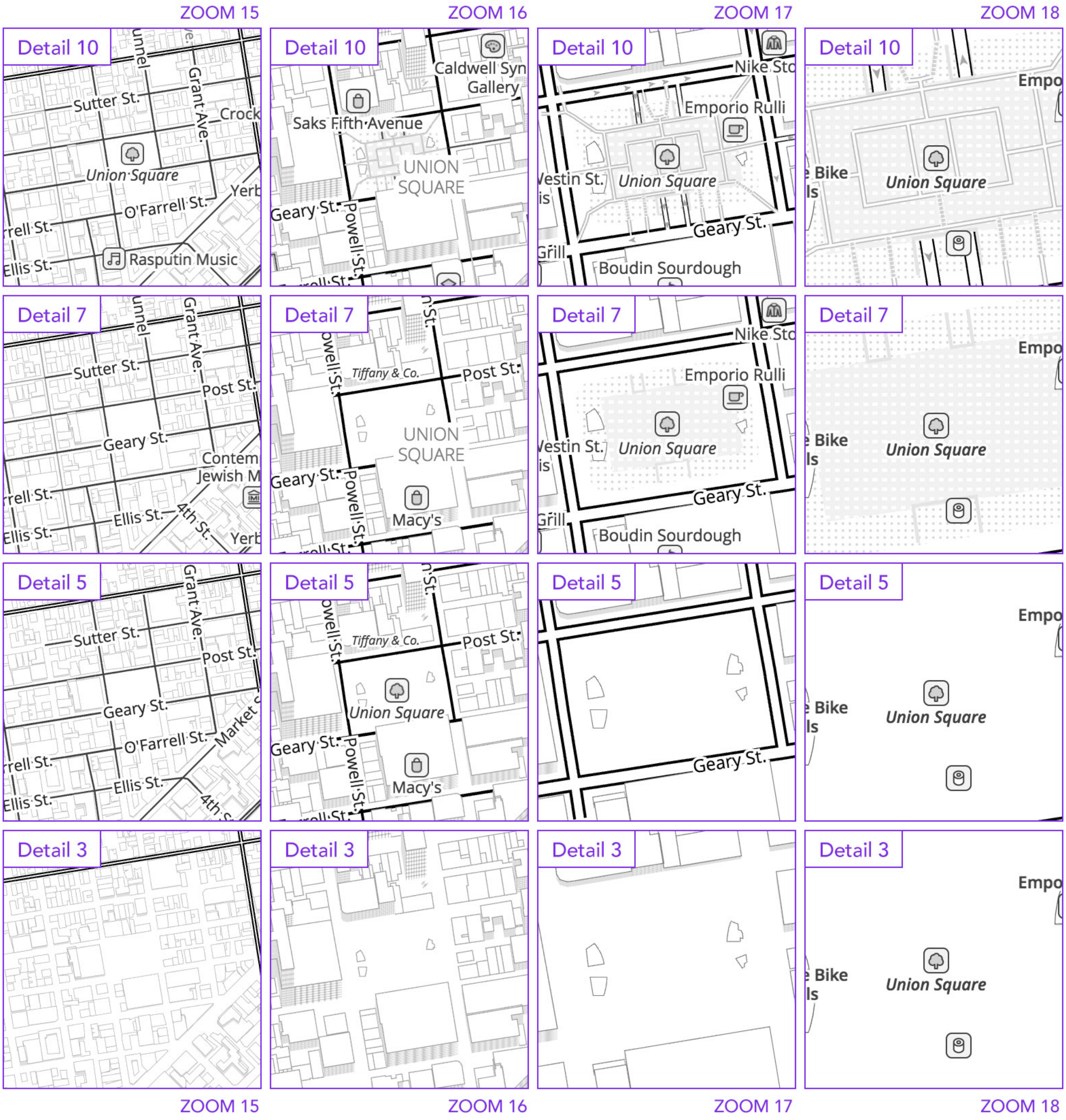
Eastern Europe at zoom-5 — detail-0 to detail-11 — Blue-Gray Refill Theme. ***View Map*** *(https://mapzen.com/products/maps/refill/detail/?detail=10&label=10&color=blue-gray#lat=43.934&lng=21.859&z=5)*

See the progression of water bodies and water labels around Växjö in Sweden at zoom level 10. Larger water bodies appear first and labels follow as you go up the levels of detail. At **detail-11** all water bodies are shown regardless of size.


Detail 10
ren
Helgasjön
Toftasjön
Alvesta
Växjö
Salen
Årydssjön
Tegnabysjön
Vederslövssjön
Lessebo
Rottnen
Åsnen
Viren

Växjö in Sweden at zoom-10 — detail-11 to detail-0 — Blue-Gray Refill Theme. ***View Map (https://mapzen.com/products/maps/refill/detail/?detail=10&label=10&color=blue-gray#lat=56.7749&lng=14.8212&z=10)***

Although it's clear-cut as to what's visible at some levels, each level of detail is never static. All levels of detail show more content accordingly as you zoom in. View Union Square in San Francisco below at **detail 3**, **5**, **7** and **10** at zoom levels 15 to 18.


ZOOM 15
ZOOM 16
ZOOM 17
ZOOM 18
Detail 10
Detail 7
Detail 5
Detail 3
Sutter St.
Grant Ave.
Union Square
O'Farrell St.
Ellis St.
Rasputin Music
Post St.
Geary St.
Contemporary Jewish Museum
4th St.
Market St.
Caldwell Snyder Gallery
Saks Fifth Avenue
UNION SQUARE
Powell St.
Tiffany & Co.
Macy's
Nike Store
Emporio Rulli
Westin St.
Boudin Sourdough
Grill
Bike
Empo

Union Square in San Francisco — Original Refill Style. ***View Map (https://mapzen.com/products/maps/refill/detail/?detail=10&label=5&color=black#lat=37.78784&lng=-122.40717&z=17)***

Label Density

You now can adjust the density of labels with label levels, and mix your labels with detail levels as well. Label levels are split up in the same way as detail: 12 steps with **0** for no labels and **11** for *"display all"*.

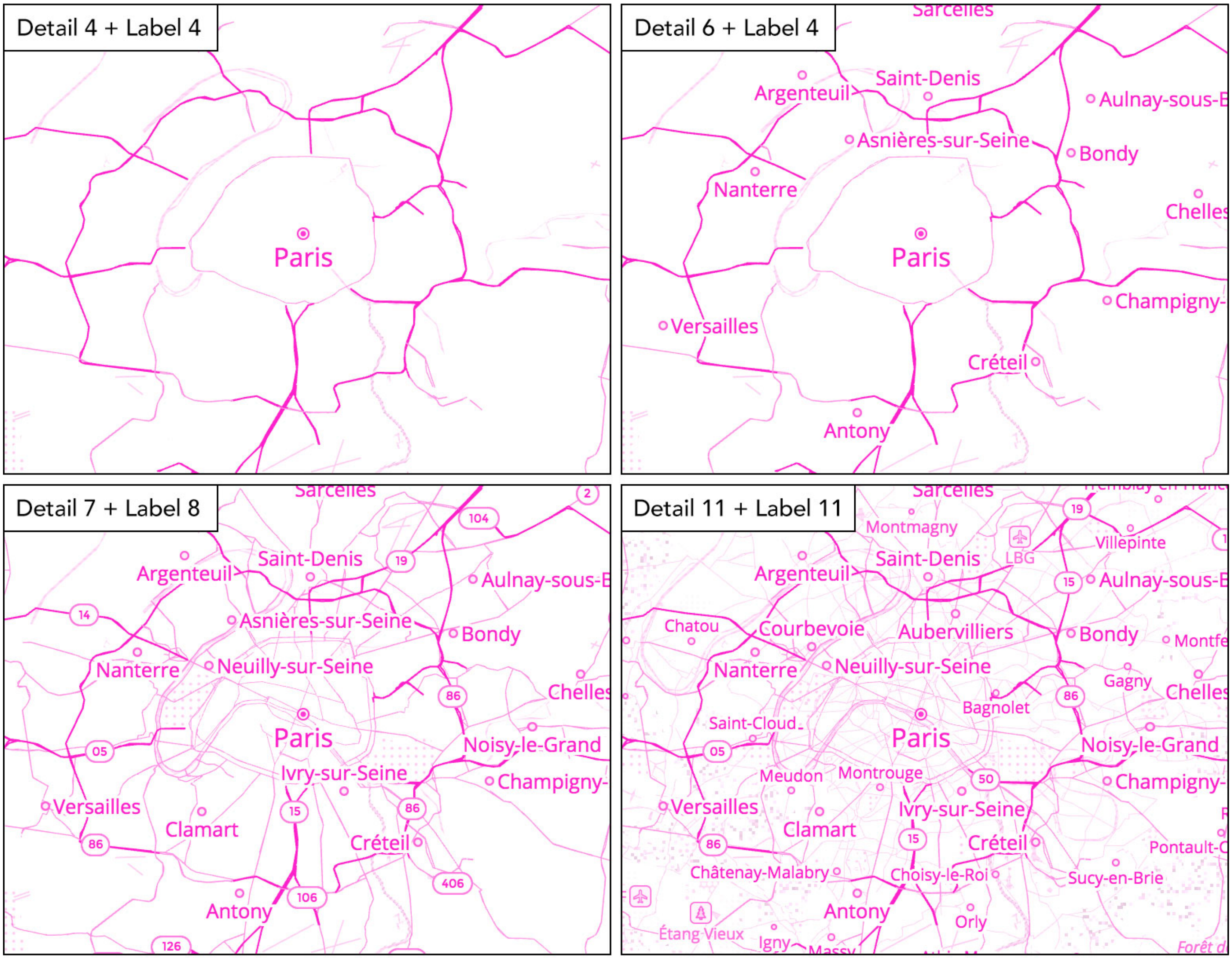
In each label step we adjust how many features of a given kind are eligible for display (by using the `min_zoom` filter discussed in the 'levels of detail' section above) and use Tangram's **buffer (https://mapzen.com/documentation/tangram/draw/#buffer)** property to adjust the spacing between displayed labels.

```
pois:
filter: function() { return feature.min_zoom <= ($zoom - 2); }
draw:
    colorized-icons:
        buffer: 48px

# new for this labels variant
poi_labels-z17:
    filter: { $zoom: [17,18,19] }
    draw:
        colorized-icons:
            buffer: 32px

# new for this labels variant
poi_labels-z19:
    filter: { $zoom: { min: 20 } }
    draw:
        colorized-icons:
            buffer: 3px
```

We also disable certain kinds of features at lower label steps by category.

City Labels

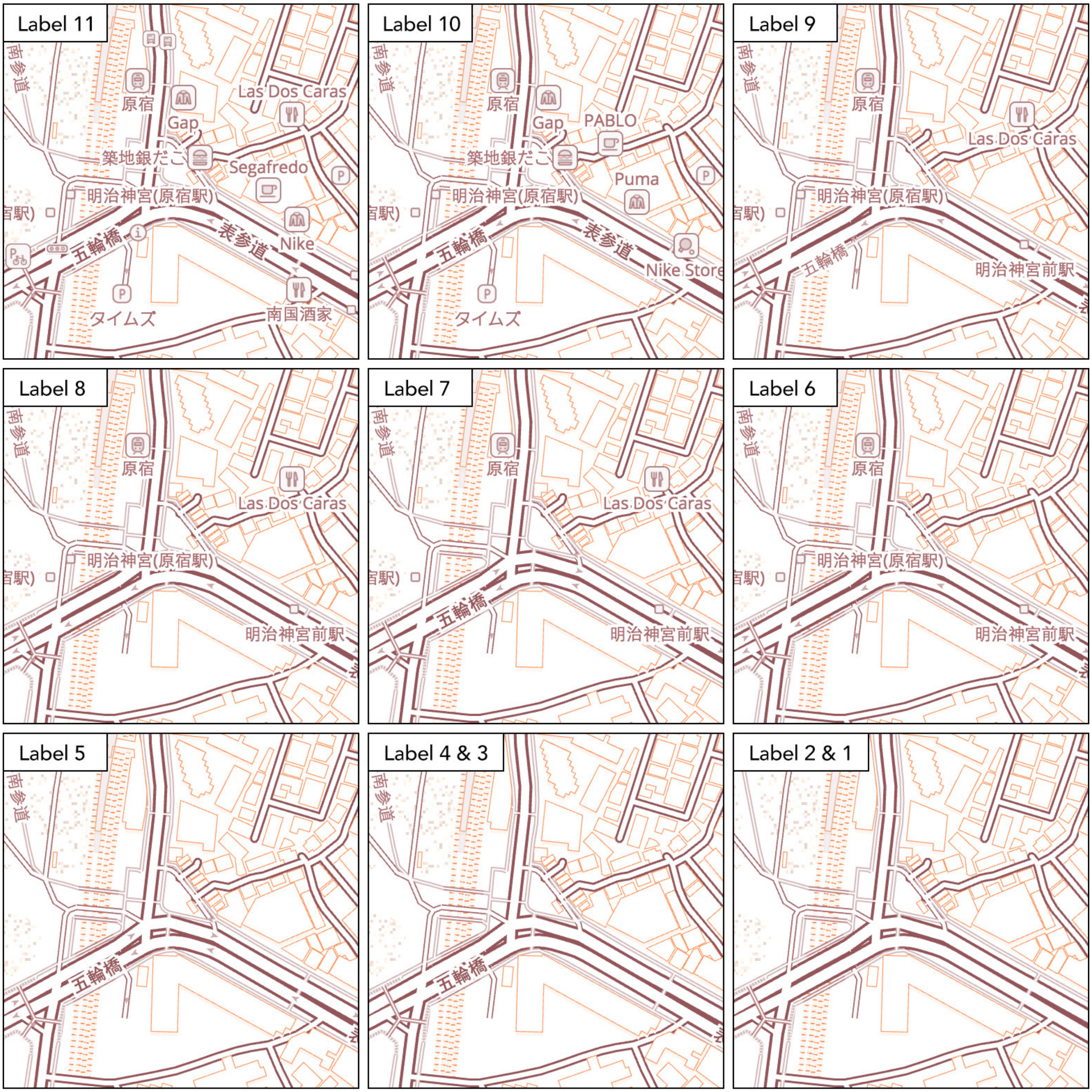
Here are a few example combinations of levels of **detail** and **label** steps of Paris at zoom level 10.


Detail 4 + Label 4
Paris
Detail 6 + Label 4
Sarcelles
Saint-Denis
Argenteuil
Aulnay-sous-B
Asnières-sur-Seine
Bondy
Nanterre
Chelles
Paris
Champigny-
Versailles
Créteil
Antony
Detail 7 + Label 8
Sarcelles
Saint-Denis
Argenteuil
Aulnay-sous-B
Asnières-sur-Seine
Bondy
Nanterre
Neuilly-sur-Seine
Chelles
Paris
Noisy-le-Grand
Ivry-sur-Seine
Champigny-
Versailles
Clamart
Créteil
Antony
2
104
19
14
86
05
15
86
86
406
106
126
Detail 11 + Label 11
Sarcelles
Montmagny
Saint-Denis
LBG
Villepinte
Argenteuil
Aulnay-sous-B
Chatou
Courbevoie
Aubervilliers
Bondy
Montfe
Nanterre
Neuilly-sur-Seine
Gagny
Chelles
Bagnolet
Saint-Cloud
Paris
Noisy-le-Grand
Meudon
Montrouge
Champigny-
Versailles
Ivry-sur-Seine
Clamart
Pontault-C
Créteil
Châtenay-Malabry
Choisy-le-Roi
Sucy-en-Brie
Antony
Orly
Étang Vieux
Igny
Massy
Forêt d
19
15
86
05
50
86
15

Paris detail and label combinations — Pink Refill Theme. ***View Map (https://mapzen.com/products/maps/refill/detail/?detail=10&label=5&color=pink#lat=48.8435&lng=2.3174&z=10)***

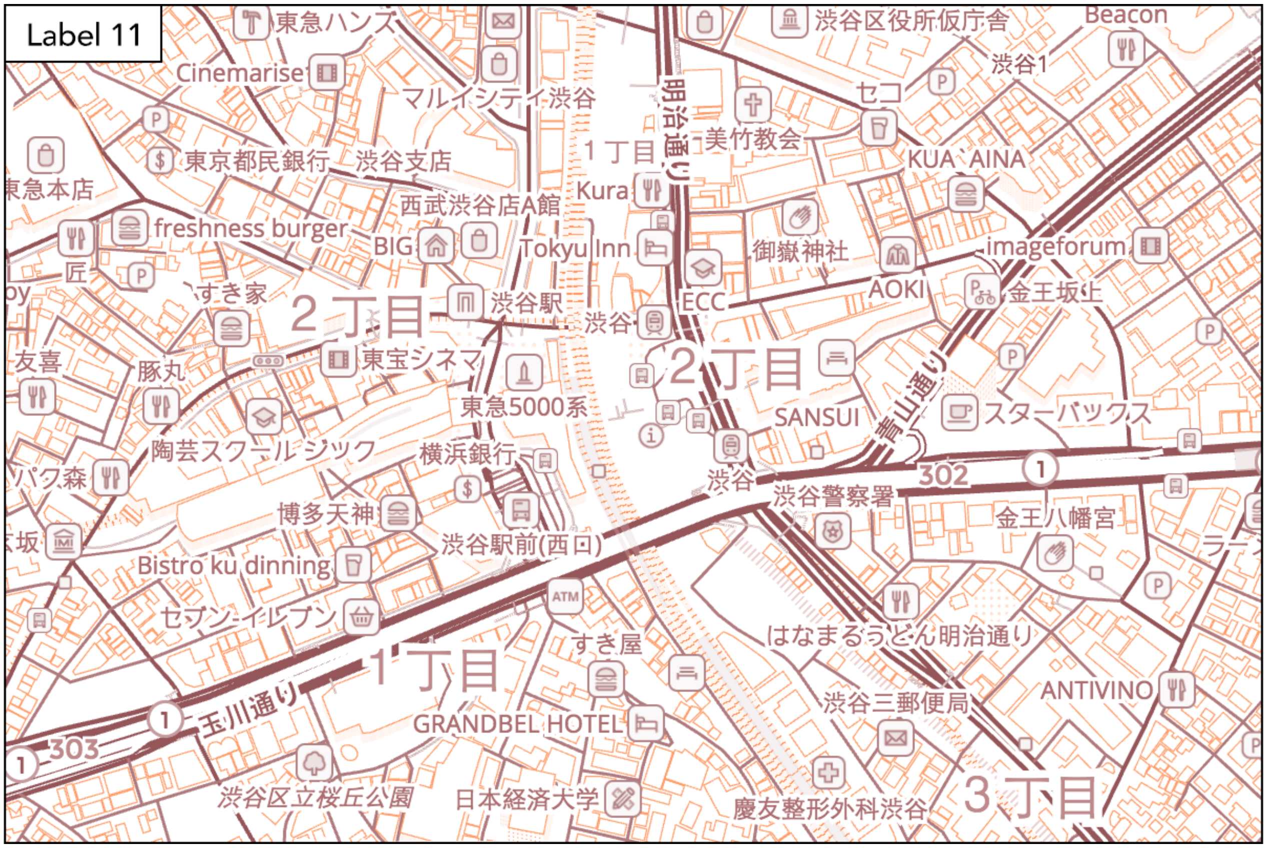
Points of Interest

See the progression of label levels with points of interest at zoom level 17. Below is a view of Meiji Jingu station in Tokyo.


Label 11
南参道
原宿
Las Dos Caras
Gap
築地銀だこ
Segafredo
明治神宮(原宿駅)
五輪橋
表参道
Nike
タイムズ
南国酒家
Label 10
PABLO
Puma
Nike Store
Label 9
明治神宮前駅
Label 8
Label 7
Label 6
Label 5
Label 4 & 3
Label 2 & 1

Meiji Jingu Station in Tokyo at zoom-17 — detail-10 + label-11 to label-1 — Brown-Orange Refill Theme.
View Map (https://mapzen.com/products/maps/refill/detail/?detail=10&label=10&color=brown-orange#lat=35.66953&lng=139.70297&z=17)

See Shibuya Station in Tokyo at zoom level 16 for another view.


Label 11
東急ハンズ
渋谷区役所仮庁舎
Beacon
Cinemarise
マルイシティ渋谷
渋谷1
明治通り
美竹教会
セコ
東京都民銀行
渋谷支店
1丁目
KUA`AINA
東急本店
西武渋谷店A館
Kura
freshness burger
BIG
Tokyu Inn
御嶽神社
imageforum
匠
すき家
2丁目
渋谷駅
渋谷
ECC
AOKI
金王坂上
友喜
豚丸
東宝シネマ
2丁目
東急5000系
SANSUI
青山通り
スターバックス
陶芸スクール ジック
横浜銀行
302
1
パク森
渋谷
渋谷警察署
金王八幡宮
博多天神
渋谷駅前(西口)
Bistro ku dinning
セブン-イレブン
ATM
すき屋
はなまるうどん明治通り
1丁目
玉川通り
渋谷三郵便局
ANTIVINO
GRANDBEL HOTEL
303
1
渋谷区立桜丘公園
日本経済大学
慶友整形外科渋谷
3丁目

Shibuya Station in Tokyo at zoom-16 — detail-10 + label-11 to label-1 — Brown-Orange Refill Theme.
View Map (https://mapzen.com/products/maps/refill/detail/?detail=10&label=10&color=brown-orange#lat=35.6582&lng=139.7025&z=16)

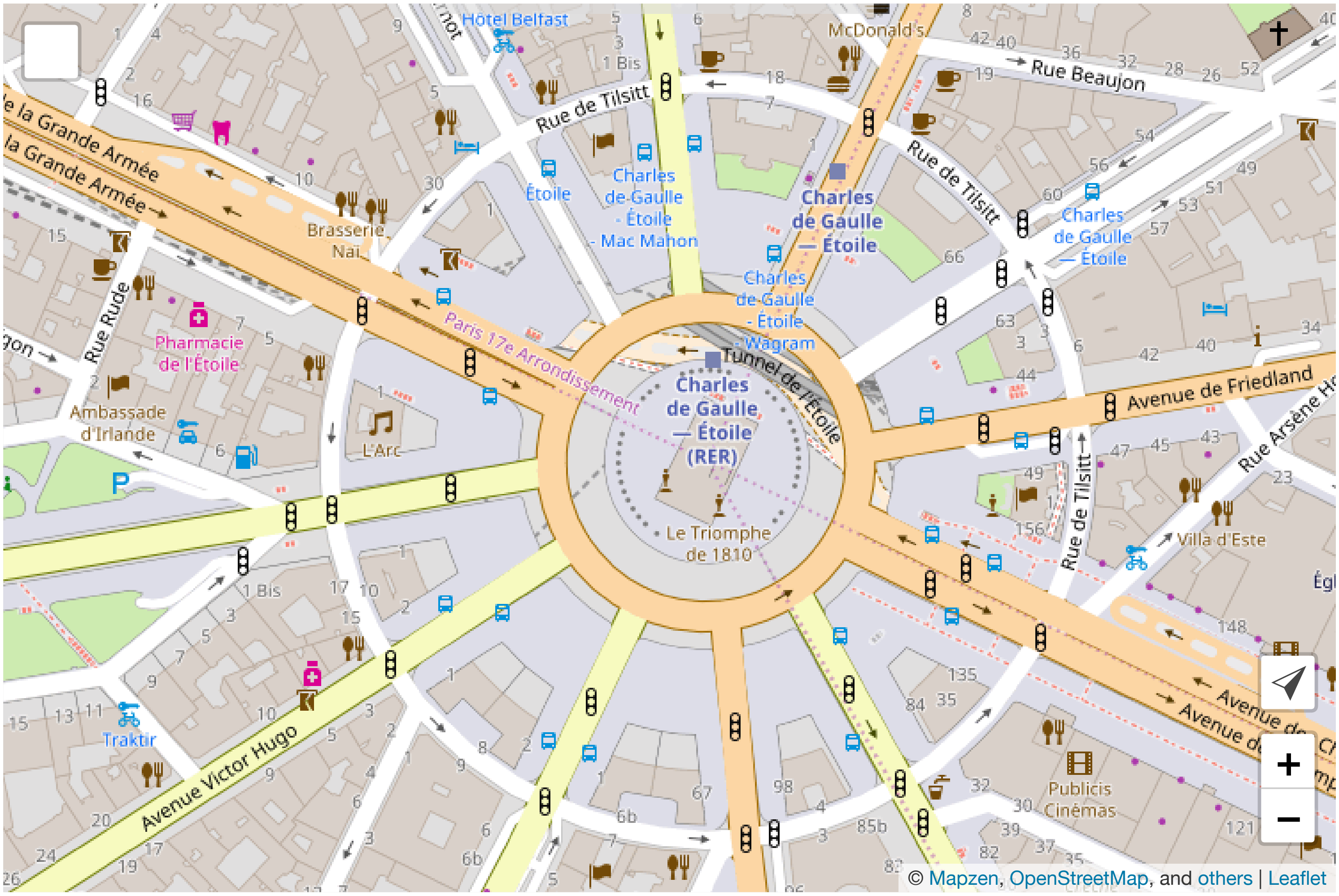
Migration

Starting with version 8 of Refill released today we recommend always importing the default `refill-style` and then adding or removing labels by importing a **label** theme. Examples are provided **below**.

Mapping of Refill label variants in version 7 and earlier releases to new theme-based label steps:

Refill label variant	label
`refill-style` *(default)*	5
`refill-style-no-labels`	0
`refill-style-more-labels`	10

The two named Refill label variants `refill-style-no-labels` or `refill-style-more-labels` available in version 7 and earlier are no longer supported directly – you need to upgrade and import the theme `label-5` or `label-10` .


Hotel Belfast
McDonald's
Rue Beaujon
Rue de Tilsitt
de la Grande Armée
la Grande Armée
Brasserie Nai
Étoile
Charles de Gaulle - Étoile - Mac Mahon
Charles de Gaulle — Étoile
Charles de Gaulle - Étoile - Wagram
Rue Rude
Pharmacie de l'Étoile
Ambassade d'Irlande
Paris 17e Arrondissement
Tunnel de l'Étoile
Charles de Gaulle — Étoile (RER)
L'Arc
Le Triomphe de 1810
Avenue de Friedland
Rue Arsène Ho
Villa d'Este
Traktir
Avenue Victor Hugo
Publicis Cinémas
Avenue de
Avenue d
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

(These maps are interactive! ***Open full screen ➶ (https://mapzen.com/products/maps/refill/detail/)*** *)*

Mix and match Refill themes

It's easy to set a theme for **color**, level of **detail**, or **label** steps after importing Refill in Tangram – just import the basemap and add a few lines! The Cartography **docs (https://mapzen.com/documentation/cartography/themes/#detail)** include the following examples.

NOTE: Order matters when importing Tangram YAML files – first import the basemap style, then import themes. You'll also need to fully qualify the URL for each import as imports are relative to the Tangram scene file's base URL (not the first import's base URL).

Set the required **Mapzen API key (https://mapzen.com/documentation/overview/#developer-accounts-and-api-keys)** to access vector tiles and import one of the theme colors:

Example **Tangram YAML** usage:

```
import:
    # basemap style
    - https://mapzen.com/carto/refill-style/8/refill-style.zip
    # modify the basemap via themes
    - https://mapzen.com/carto/refill-style/8/themes/detail-8.zip
    - https://mapzen.com/carto/refill-style/8/themes/label-10.zip
    - https://mapzen.com/carto/refill-style/8/themes/color-pink-yellow.zip
    # other imports to taste...

global:
    sdk_mapzen_api_key: mapzen-xxxxxx
    # are 3d buildings casting too dark a shadow on your map?
    # show 2d building footprints instead (optional)
    sdk_building_extrude: false
```

*NOTE: As Mapzen's basemaps are still in active development we recommend pegging an import to a specific **MAJOR** version, e.g.:* `8` *, so you enjoy any minor and patch updates but are ensured of stable named scene elements.*

Example **Tangram JS** usage:

```
var layer = Tangram.leafletLayer({
        scene: {
          import: [
            'https://mapzen.com/carto/refill-style/8/refill-style.zip',
            'https://mapzen.com/carto/refill-style/8/themes/detail-8.zip',
            'https://mapzen.com/carto/refill-style/8/themes/label-10.zip',
            'https://mapzen.com/carto/refill-style/8/themes/color-pink-yellow.zip'],
          global: { 'sdk_mapzen_api_key': 'mapzen-xxxxxx', 'sdk_building_extrude': 'false
        attribution: '<a href="https://mapzen.com/tangram" target="_blank">Tangram</a>, &
    });
```

If you're using **mapzen.js (https://github.com/mapzen/mapzen.js)**:

```
var map = L.Mapzen.map('map', {
  center: [40.8041, -124.1506],
  zoom: 15,
  maxZoom: 20,
  tangramOptions: {
    scene: {
      import: [
        'https://mapzen.com/carto/refill-style/8/refill-style.zip',
        'https://mapzen.com/carto/refill-style/8/themes/detail-8.zip',
        'https://mapzen.com/carto/refill-style/8/themes/label-10.zip',
        'https://mapzen.com/carto/refill-style/8/themes/color-pink-yellow.zip'],
      global: { 'sdk_mapzen_api_key': 'mapzen-xxxxxx', 'sdk_building_extrude': 'false' }
    } }
});
```

Tangram ES, **Mapzen JS**, and **Mapzen Android and iOS SDK** support is coming soon.

· 24 July 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

Redesigning and Rebuilding the Who's On First website

whosonfirst (/tag/whosonfirst) data (/tag/data) interns (/tag/interns)

My name is **Scott (http://scottdombkowski.com/)** and I am an intern on the Mapzen data team this summer. I am also pursuing a graduate degree at the **Carnegie Mellon School of Design (http://design.cmu.edu/)**.

At Mapzen, I am tasked with **improving how Who's On First (WOF) is presented to the world and increasing its accessibility to a wider range of users (https://whosonfirst.mapzen.com/interns/#2017-ux)**.

For the majority of my first three weeks at Mapzen, I have focused on the redesign and the rebuilding of the **WOF website (https://whosonfirst.mapzen.com/)**. (For those who aren't familiar with the current website, here's a **full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstCurrentState.png)**)

The website currently contains:

- **an introduction/explanation of Who's On First (https://whosonfirst.mapzen.com)** (What is WOF?, What state WOF is in?)
- **a section with links to WOF documentation (https://whosonfirst.dev.mapzen.com/#documentation)**
- **a section with links to WOF tools (https://whosonfirst.mapzen.com/#examples)** (ex: WOF spelunker (a tool to browse WOF data))
- **a section with links to the thinking surrounding the project (https://whosonfirst.mapzen.com/#theory)** (What is the purpose of WOF?, How are WOF places organized?)
- **a section with links to access WOF data and explanations on how the data is organized (https://whosonfirst.mapzen.com/#data)** (Where does the data live?, Why is the data organized that way?)
- **a section with links to WOF sources (https://whosonfirst.mapzen.com/#sources)** (human-friendly and JSON formats)

- **a section dedicated to licensing and how to credit WOF and other projects properly (https://whosonfirst.mapzen.com/#license)**

Goals

After gaining a better understanding of the website and the project itself, I developed four goals to help guide the redesign:

- Provide an overview of the project showing the entirety of the thought put into the project, while not being so overwhelming as to scare potential users off.
- Provide the tools to immediately utilize WOF.
- Provide the tools for deeper exploration into WOF.
- Get individuals excited about the project.

Users

I also began to think about the users of the website. Are they all similar? Are they coming to the project for different reasons? What kind of backgrounds and experiences do they bring with them?

I concluded that the majority of these users will come from three different groups.

1. *Users actively working on a project that could benefit from WOF.* They are not stumbling upon the project, they have come to the website with a specific purpose. They have either used WOF before or have background knowledge about WOF. They have some technical knowledge.
2. *Users not actively working on a project that could benefit from using WOF.* These users are interested in WOF, but don't have an immediate need for it. They are interested in learning about the project, but are not looking for in-depth knowledge. They may or may not have technical knowledge.
3. *Users who stumble upon WOF.* They come to the project without preconceptions. They may never have a use for WOF, but may spend some time on the page if they find it interesting.

Ideation

With established goals and thoughts concerning website users, new questions began popping up.

- How can we most effectively allow for understanding of WOF?
- What are the most effective ways of visualizing WOF?
- Can understanding of WOF be acquired through some sort of interaction?
- How can users directly interact with the data?
- How can we replicate a user's interaction with WOF outside the website within the website?
- Can we utilize the website's content to create a dialog that gets the attention of a user while also showing the depth of the project?
- How can we visualize that this project is a continuous work in progress?

With these questions and others in mind, I searched for answers moving from lower fidelity to higher fidelity prototypes.

Note: Throughout the design process, I have been using the ***Mapzen Styleguide (https://mapzen.com/common/styleguide/)*** *so that anything I was designing or building would align with the Mapzen visual style and other Mapzen product pages.*

Prototyping

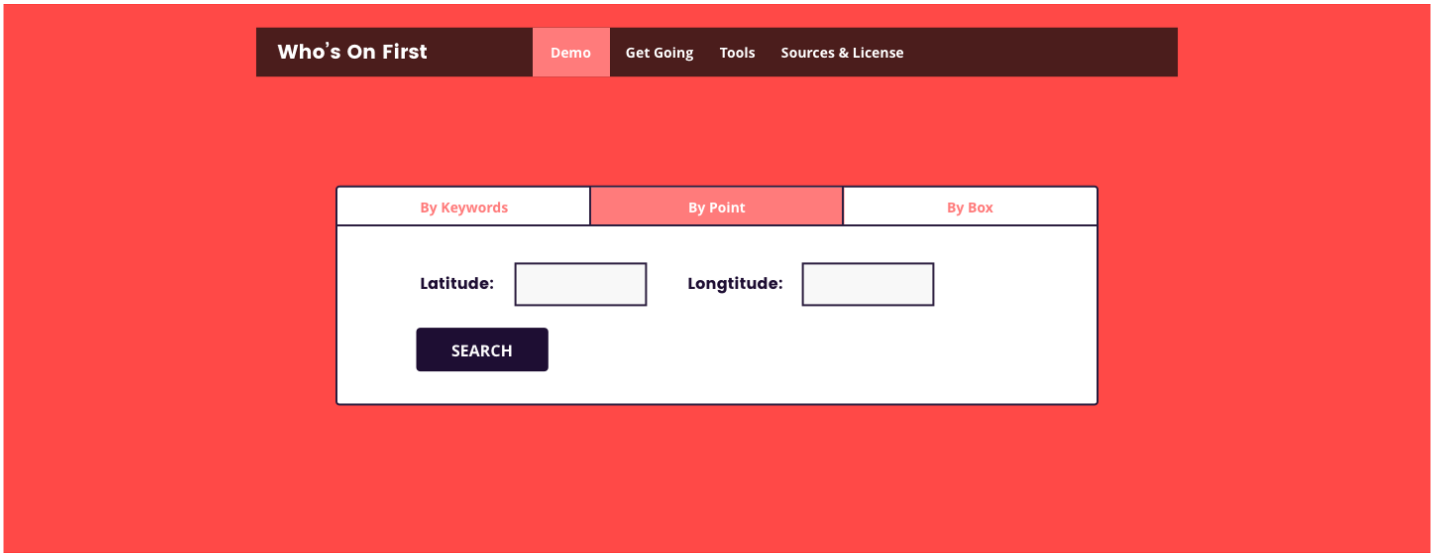
One of the first things I focused on was how users directly interact with WOF data. Do we create some sort of interactive component where individuals can immediately interact and get a sense of the data made available by WOF? If so, what powers that component? One feature that WOF provides is an **API (https://mapzen.com/documentation/wof/)**. We talked about this **a few months back (https://mapzen.com/blog/whosonfirst-api/)**:

> The API provides programmatic access for you and your robots to all the Who's On First data. You can query individual places and their relations, look for concordances and perform basic spatial queries. You can also use the API to query things like all the different placetypes or sources for data.

While some of the **API queries (https://mapzen.com/documentation/wof/methods/)** require background knowledge, a number of them could be understood by a first-time user, like **getByLatLon (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getByLatLon)**.

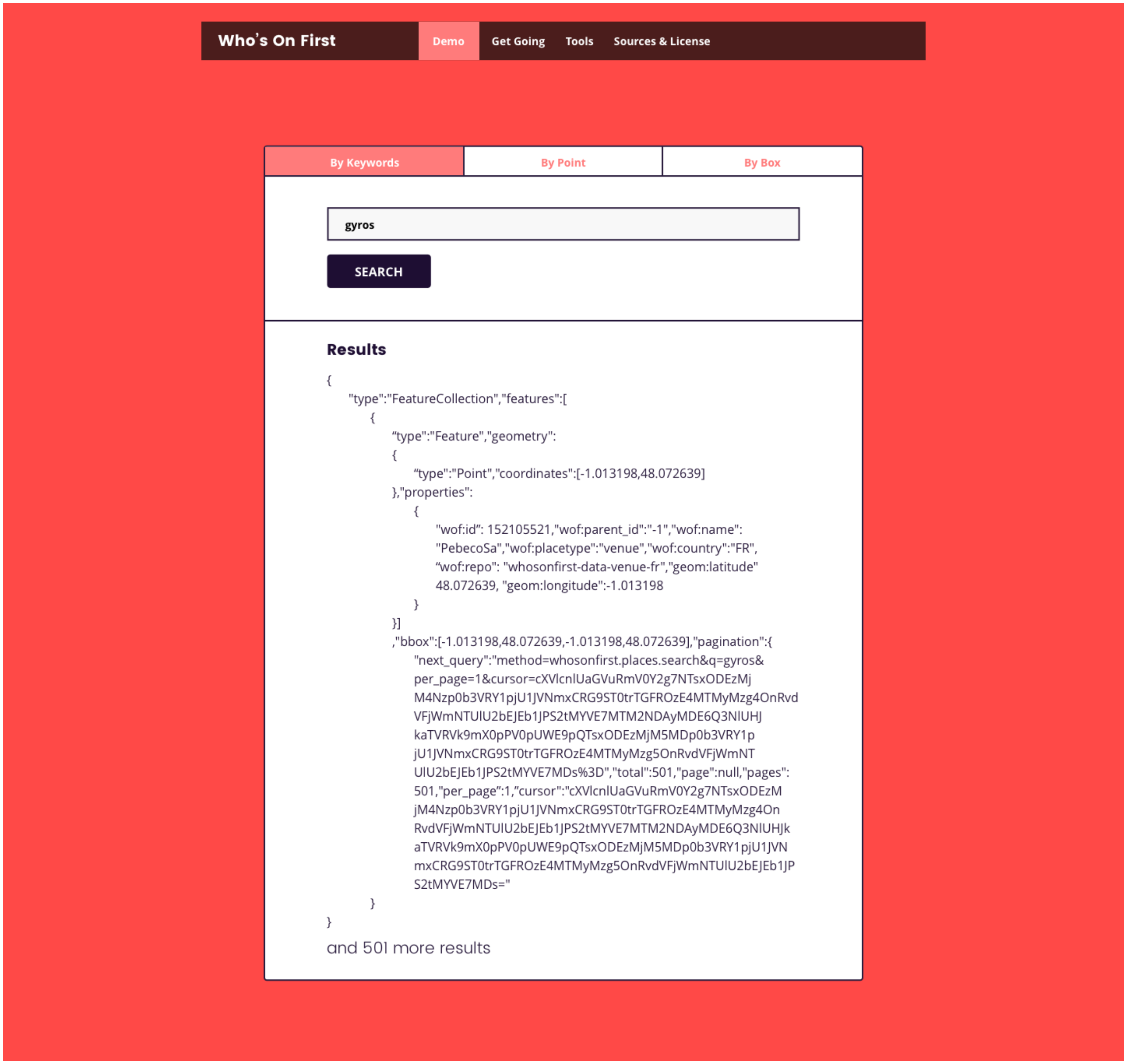
Knowing that I could use the **API (https://mapzen.com/documentation/wof/)**, I explored different ways the **API methods (https://mapzen.com/documentation/wof/methods/)** could be visualized.

Could we create a card with tabs that would display specific fields based on the tab selected?


Who's On First
Demo
Get Going
Tools
Sources & License
By Keywords
By Point
By Box
Latitude:
Longtitude:
SEARCH

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock7Small.png)

Results would appear below the tabs and fields.



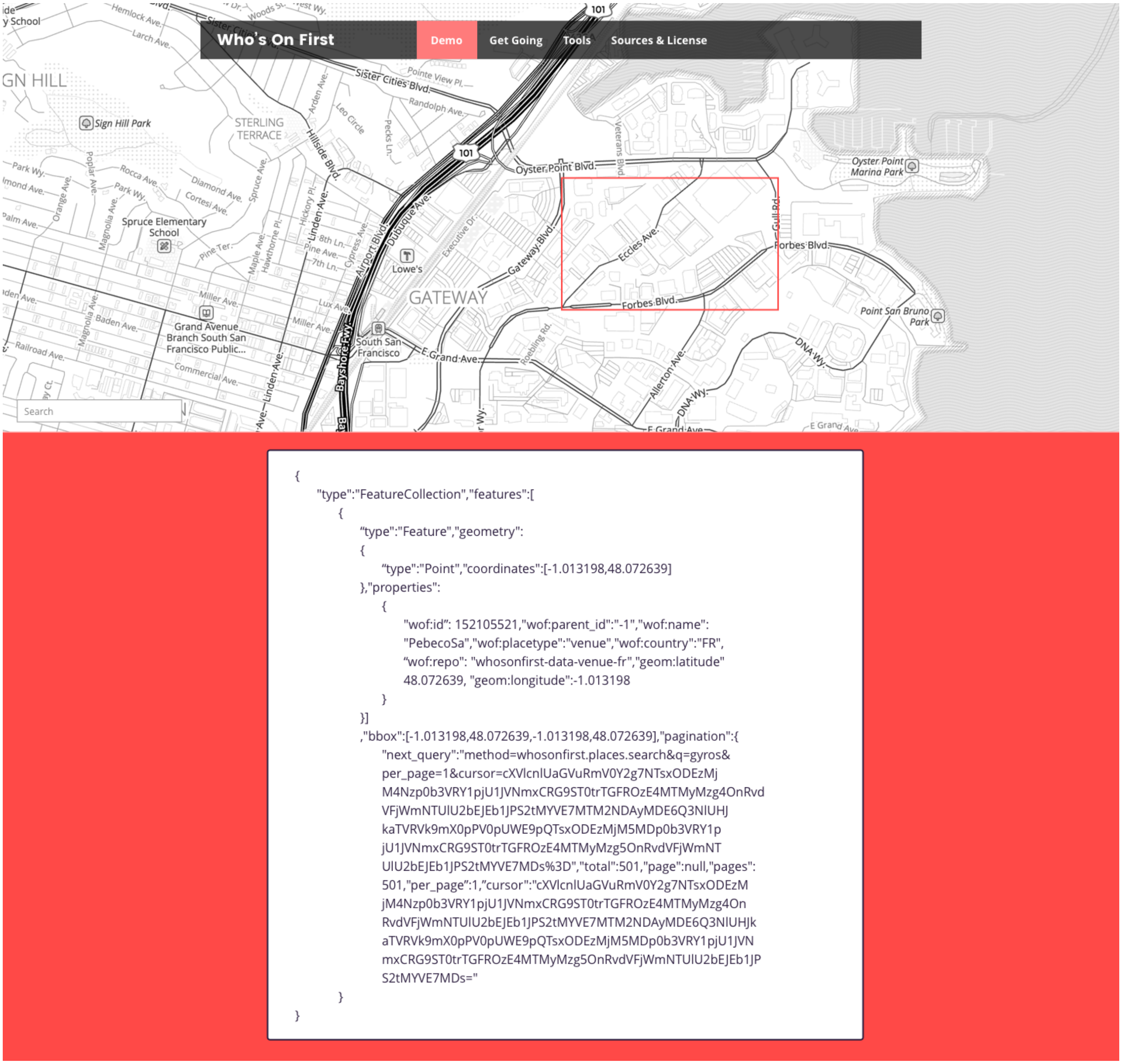
Results

```
{
    "type":"FeatureCollection","features":[
        {
            “type":"Feature","geometry":
            {
                “type":"Point","coordinates":[-1.013198,48.072639]
            },"properties":
                {
                    "wof:id”: 152105521,"wof:parent_id":"-1","wof:name":
                    "PebecoSa","wof:placetype":"venue","wof:country":"FR",
                    “wof:repo": "whosonfirst-data-venue-fr","geom:latitude"
                    48.072639, "geom:longitude":-1.013198
                }
            }]
            ,"bbox":[-1.013198,48.072639,-1.013198,48.072639],"pagination":{
                "next_query":"method=whosonfirst.places.search&q=gyros&
                per_page=1&cursor=cXVlcnlUaGVuRmV0Y2g7NTsxODEzMj
                M4Nzp0b3VRY1pjU1JVNmxCRG9ST0trTGFROzE4MTMyMzg4OnRvd
                VFjWmNTUlU2bEJEb1JPS2tMYVE7MTM2NDAyMDE6Q3NlUHJ
                kaTVRVk9mX0pPV0pUWE9pQTsxODEzMjM5MDp0b3VRY1p
                jU1JVNmxCRG9ST0trTGFROzE4MTMyMzg5OnRvdVFjWmNT
                UlU2bEJEb1JPS2tMYVE7MDs%3D","total":501,"page":null,"pages":
                501,"per_page”:1,”cursor":"cXVlcnlUaGVuRmV0Y2g7NTsxODEzM
                jM4Nzp0b3VRY1pjU1JVNmxCRG9ST0trTGFROzE4MTMyMzg4On
                RvdVFjWmNTUlU2bEJEb1JPS2tMYVE7MTM2NDAyMDE6Q3NlUHJk
                aTVRVk9mX0pPV0pUWE9pQTsxODEzMjM5MDp0b3VRY1pjU1JVN
                mxCRG9ST0trTGFROzE4MTMyMzg5OnRvdVFjWmNTUlU2bEJEb1JP
                S2tMYVE7MDs="
        }
}
```

and 501 more results

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock2Small.png)

Or we could create a map that would allow for interaction? A user could select a specific point or draw a box on the map and results based on that point or rectangle would appear.


Who's On First
Demo
Get Going
Tools
Sources & License
Search
GATEWAY
STERLING TERRACE
Sign Hill Park
Spruce Elementary School
Grand Avenue Branch South San Francisco Public...
South San Francisco
Lowe's
Oyster Point Marina Park
Point San Bruno Park
Oyster Point Blvd.
Forbes Blvd.
Eccles Ave.
Gateway Blvd.
E Grand Ave.
Allerton Ave.
DNA Wy.
Bayshore Fwy.
101

```
{
    "type":"FeatureCollection","features":[
        {
            "type":"Feature","geometry":
            {
                "type":"Point","coordinates":[-1.013198,48.072639]
            },"properties":
                {
                    "wof:id": 152105521,"wof:parent_id":"-1","wof:name":
                    "PebecoSa","wof:placetype":"venue","wof:country":"FR",
                    "wof:repo": "whosonfirst-data-venue-fr","geom:latitude"
                    48.072639, "geom:longitude":-1.013198
                }
            }]
            ,"bbox":[-1.013198,48.072639,-1.013198,48.072639],"pagination":{
                "next_query":"method=whosonfirst.places.search&q=gyros&
                per_page=1&cursor=cXVlcnlUaGVuRmV0Y2g7NTsxODEzMj
                M4Nzp0b3VRY1pjU1JVNmxCRG9ST0trTGFROzE4MTMyMzg4OnRvd
                VFjWmNTUlU2bEJEb1JPS2tMYVE7MTM2NDAyMDE6Q3NlUHJ
                kaTVRVk9mX0pPV0pUWE9pQTsxODEzMjM5MDp0b3VRY1p
                jU1JVNmxCRG9ST0trTGFROzE4MTMyMzg5OnRvdVFjWmNT
                UlU2bEJEb1JPS2tMYVE7MDs%3D","total":501,"page":null,"pages":
                501,"per_page":1,"cursor":"cXVlcnlUaGVuRmV0Y2g7NTsxODEzM
                jM4Nzp0b3VRY1pjU1JVNmxCRG9ST0trTGFROzE4MTMyMzg4On
                RvdVFjWmNTUlU2bEJEb1JPS2tMYVE7MTM2NDAyMDE6Q3NlUHJk
                aTVRVk9mX0pPV0pUWE9pQTsxODEzMjM5MDp0b3VRY1pjU1JVN
                mxCRG9ST0trTGFROzE4MTMyMzg5OnRvdVFjWmNTUlU2bEJEb1JP
                S2tMYVE7MDs="
        }
}
```

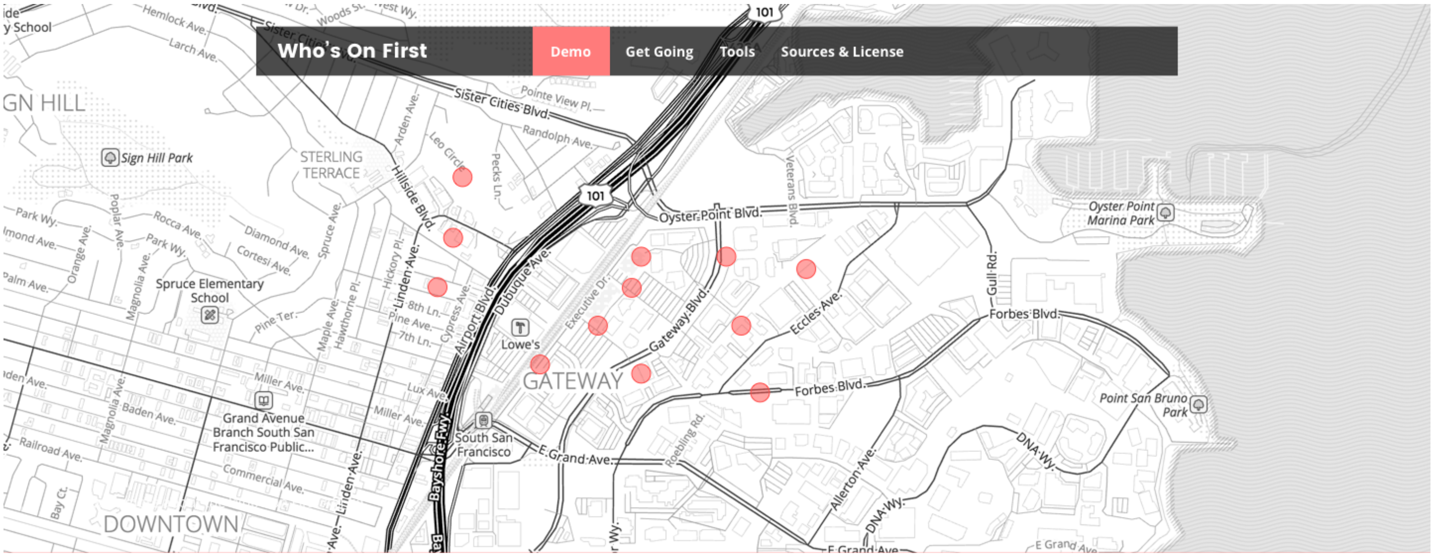
See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock10Small.png)

I had some concerns with this approach. Would the interaction interfere with the main navigation of the page? Should the first thing be an element relying on user interaction? I ultimately came to the conclusion that this first element should not rely on a user's interaction.

Initial Component

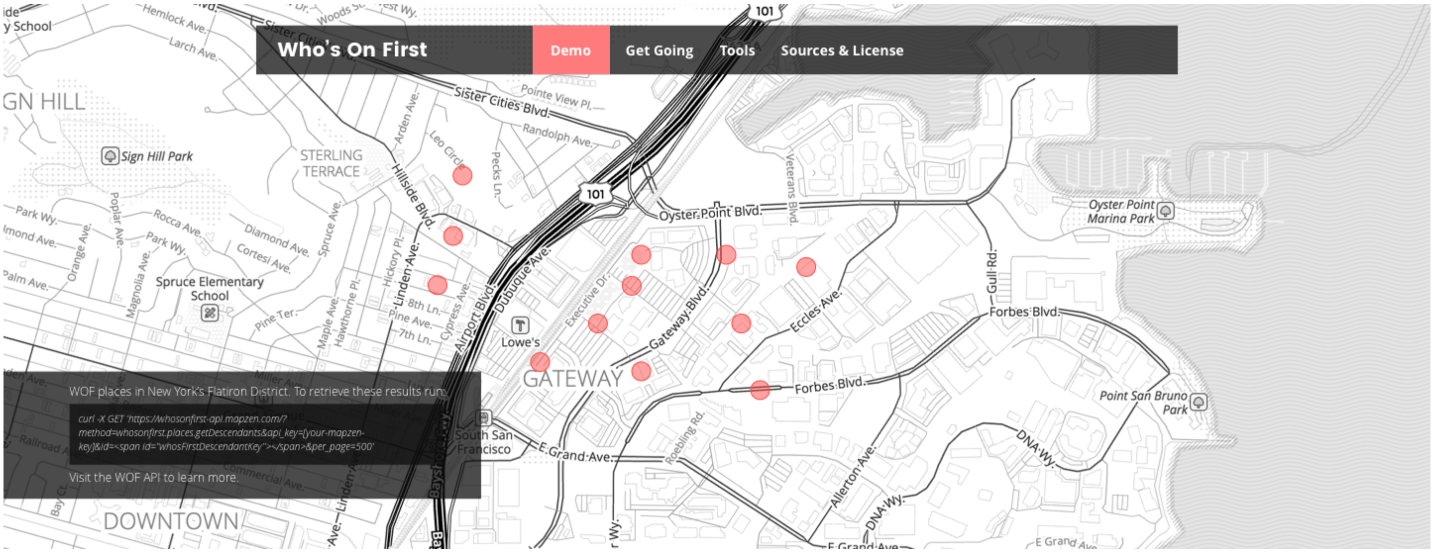
Since I didn't want the first element to rely on a user interacting with it, I created further iterations.

I knew that I wanted the element to immediately capture the user's attention and show some of the capabilities of WOF. With that in mind, I created the mockup below where users are able to see WOF places depicted by red circles.


Who's On First
Demo
Get Going
Tools
Sources & License
GATEWAY
DOWNTOWN

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock14Small.png)

I showed it to my team and they suggested including the API command powering the visualization.


Who's On First
Demo
Get Going
Tools
Sources & License
GATEWAY
DOWNTOWN

WOF places in New York's Flatiron District. To retrieve these results run:

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.getDescendants&api_key={your-mapzen-key}&id=<span id="whosFirstDescendantKey"></span>&per_page=500'
```

Visit the WOF API to learn more.

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock19Small.png)

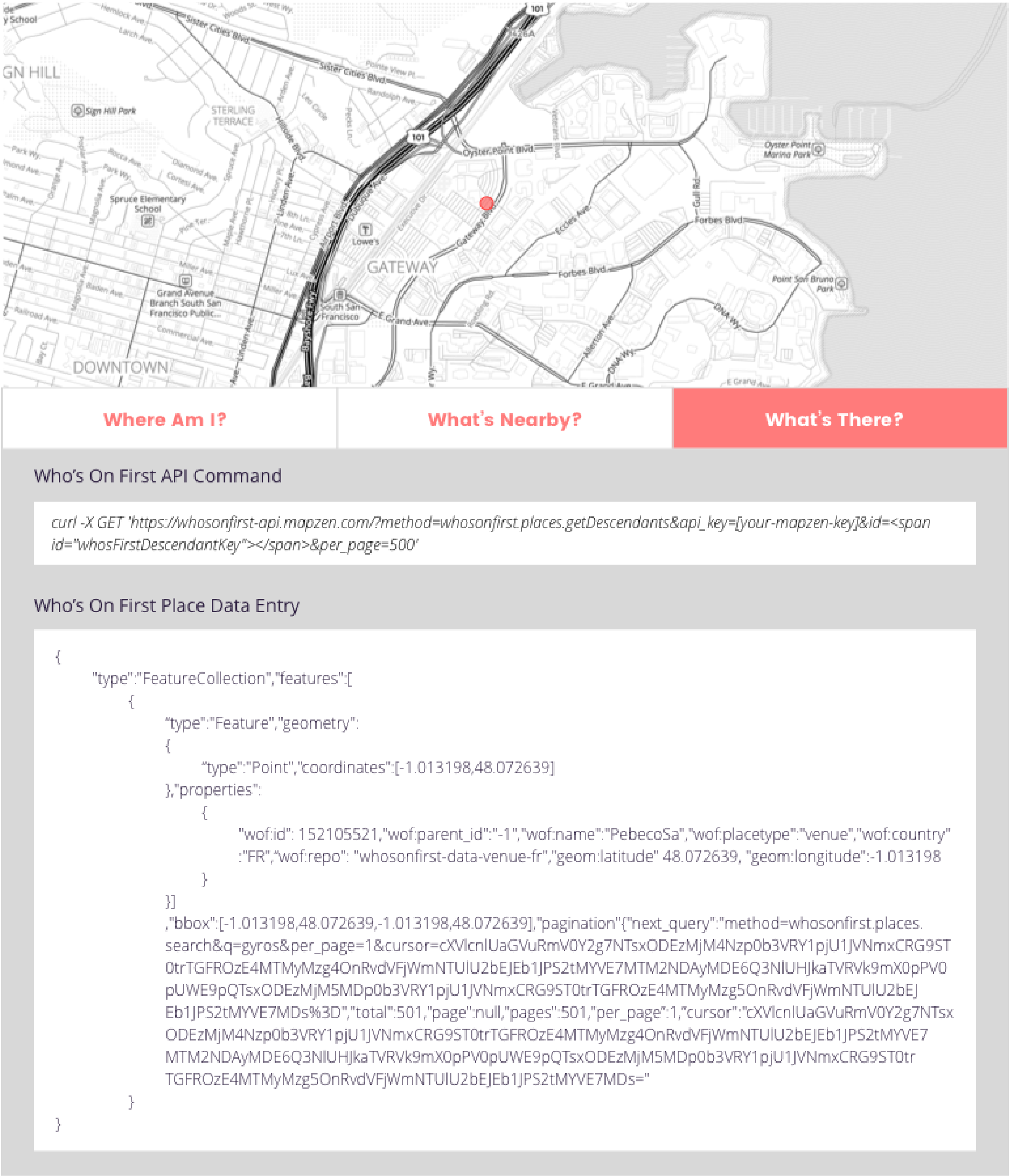
When initially building this out, I'd planned to detect a user's location and run a WOF API query based on that location. However, this can lead to **potential problems with GeoIP lookups (https://mapzen.com/blog/missing-the-point/)**. The mockup is now built to randomly select one of five locations and run a WOF API query based on that location.

Interactive Component

After building out the initial component, I still wanted to create an element that would allow for a deeper and more direct interaction with WOF. Consulting with my team about **the most important API queries (https://mapzen.com/documentation/wof/methods/#whosonfirstplaces)**, we settled on:

- **getByLatLon (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getByLatLon)**: returns WOF places intersecting a latitude and longitude
- **getNearby (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getNearby)**: returns all the WOF records near a point
- **getIntersects (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getIntersects)**: returns all the WOF places intersecting a bounding box

Using my earlier mockups as inspiration, I thought about ways a user could run these queries through an interface. This inspired another set of mockups.


GATEWAY
DOWNTOWN
Sister Cities Blvd
Oyster Point Blvd
Forbes Blvd
Oyster Point Marina Park
Point San Bruno Park
South San Francisco
Where Am I?
What's Nearby?
What's There?

Who's On First API Command

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.getDescendants&api_key=[your-mapzen-key]&id=<span
id="whosFirstDescendantKey"></span>&per_page=500'
```

Who's On First Place Data Entry

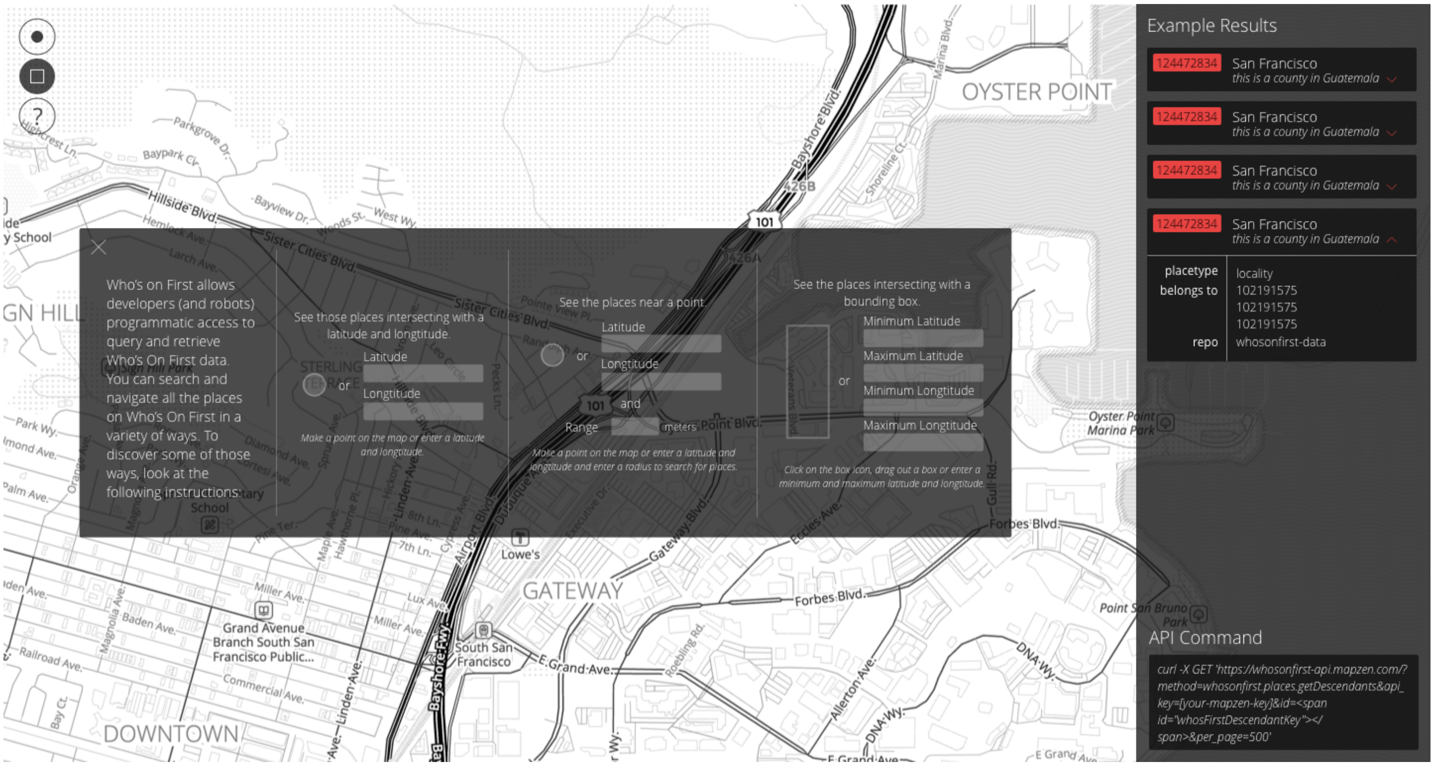
```
{
        "type":"FeatureCollection","features":[
                {
                        "type":"Feature","geometry":
                        {
                                "type":"Point","coordinates":[-1.013198,48.072639]
                        },"properties":
                                {
                                        "wof:id": 152105521,"wof:parent_id":"-1","wof:name":"PebecoSa","wof:placetype":"venue","wof:country"
                                        :"FR","wof:repo": "whosonfirst-data-venue-fr","geom:latitude" 48.072639, "geom:longitude":-1.013198
                                }
                        }]
                        ,"bbox":[-1.013198,48.072639,-1.013198,48.072639],"pagination"{"next_query":"method=whosonfirst.places.
                        search&q=gyros&per_page=1&cursor=cXVlcnlUaGVuRmV0Y2g7NTsxODEzMjM4Nzp0b3VRY1pjU1JVNmxCRG9ST
                        0trTGFROzE4MTMyMzg4OnRvdVFjWmNTUlU2bEJEb1JPS2tMYVE7MTM2NDAyMDE6Q3NlUHJkaTVRVk9mX0pPV0
                        pUWE9pQTsxODEzMjM5MDp0b3VRY1pjU1JVNmxCRG9ST0trTGFROzE4MTMyMzg5OnRvdVFjWmNTUlU2bEJ
                        Eb1JPS2tMYVE7MDs%3D","total":501,"page":null,"pages":501,"per_page":1,"cursor":"cXVlcnlUaGVuRmV0Y2g7NTsx
                        ODEzMjM4Nzp0b3VRY1pjU1JVNmxCRG9ST0trTGFROzE4MTMyMzg4OnRvdVFjWmNTUlU2bEJEb1JPS2tMYVE7
                        MTM2NDAyMDE6Q3NlUHJkaTVRVk9mX0pPV0pUWE9pQTsxODEzMjM5MDp0b3VRY1pjU1JVNmxCRG9ST0tr
                        TGFROzE4MTMyMzg5OnRvdVFjWmNTUlU2bEJEb1JPS2tMYVE7MDs="
                }
}
```

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock19Small2.png)

These brought up a new question.

Could you combine the **getByLatLong function (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getByLatLon)** with the **getNearby function (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getNearby)**? The **getByLatLong function (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getByLatLon)** is essentially equivalent to the **getNearby function (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getNearby)**, without the radius argument.

This convinced me to combine the **getByLatLong (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getByLatLon)** and **getNearby (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getNearby)** queries into one type of interaction, while the **getIntersects query (https://mapzen.com/documentation/wof/methods/#whosonfirst.places.getIntersects)** would be represented by a different interaction. They would be represented by the icons in the top left of the mockup below.



Who's on First allows developers (and robots) programmatic access to query and retrieve Who's On First data. You can search and navigate all the places on Who's On First in a variety of ways. To discover some of those ways, look at the following instructions:

See those places intersecting with a latitude and longtitude.

Make a point on the map or enter a latitude and longtitude.

See the places near a point.

Make a point on the map or enter a latitude and longtitude and enter a radius to search for places.

See the places intersecting with a bounding box.

Click on the box icon, drag out a box or enter a minimum and maximum latitude and longtitude.

Example Results

124472834 San Francisco
this is a county in Guatemala

124472834 San Francisco
this is a county in Guatemala

124472834 San Francisco
this is a county in Guatemala

124472834 San Francisco
this is a county in Guatemala

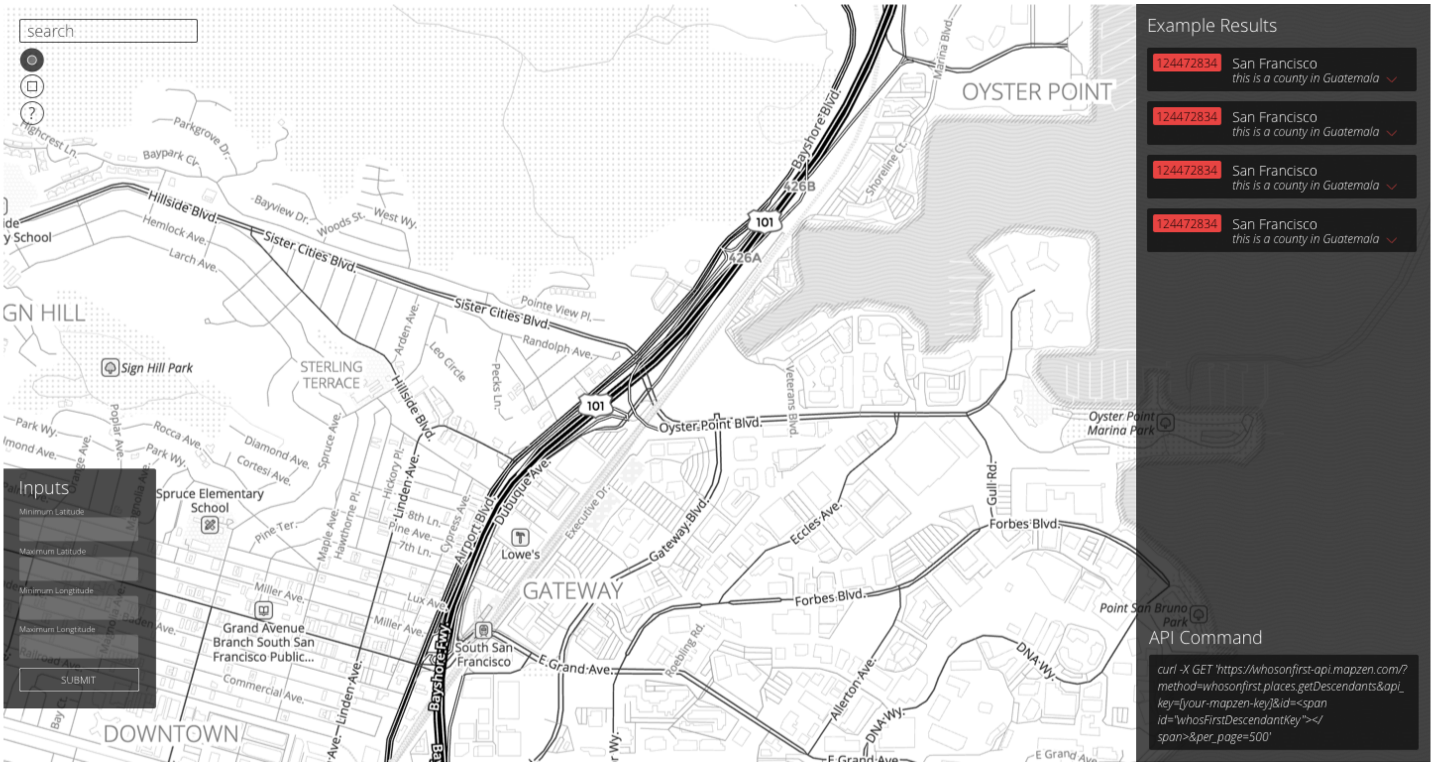
placetype	locality
belongs to	102191575 102191575 102191575
repo	whosonfirst-data

API Command

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.getDescendants&api_key=[your-mapzen-key]&id=<span id="whosFirstDescendantKey"></span>&per_page=500'
```

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/MapInteraction33.png)

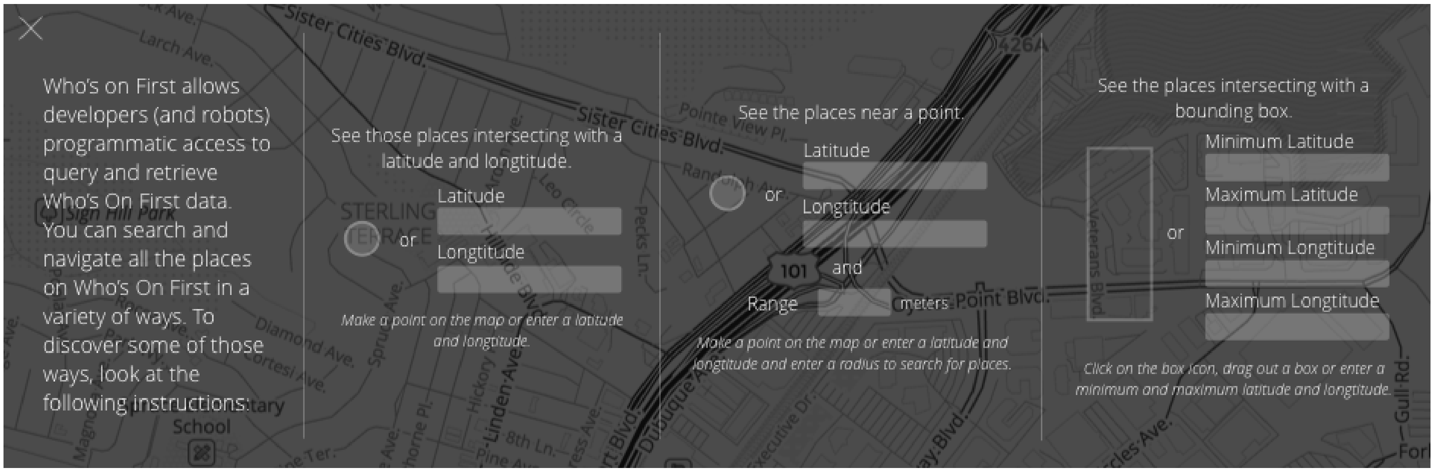
The point (dot) icon would represent the `getByLatLong` and `getNearby` interaction. When active, users would be able to select a point on the map and/or specify a radius, latitude, and longitude in a window on the bottom left. The box icon would represent the `getIntersects` interaction. When active, users would be able to draw a box on the map or specify minimum and maximum latitudes and longitudes in a window on the bottom left. The results of the those queries would be seen in the right hand column, followed by the API query run to acquire those results.


search
Inputs
Minimum Latitude
Maximum Latitude
Minimum Longtitude
Maximum Longtitude
SUBMIT
Example Results
124472834 San Francisco
this is a county in Guatemala
124472834 San Francisco
this is a county in Guatemala
124472834 San Francisco
this is a county in Guatemala
124472834 San Francisco
this is a county in Guatemala
API Command
OYSTER POINT
GATEWAY
DOWNTOWN

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.getDescendants&api_key=[your-mapzen-key]&id=<span id="whosFirstDescendantKey"></span>&per_page=500'
```

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/MapInteraction39.png)

We would also need a walkthrough to help users understand the functionality of the component.



Who's on First allows developers (and robots) programmatic access to query and retrieve Who's On First data. You can search and navigate all the places on Who's On First in a variety of ways. To discover some of those ways, look at the following instructions:

See those places intersecting with a latitude and longtitude.

Make a point on the map or enter a latitude and longtitude.

See the places near a point.

Make a point on the map or enter a latitude and longtitude and enter a radius to search for places.

See the places intersecting with a bounding box.

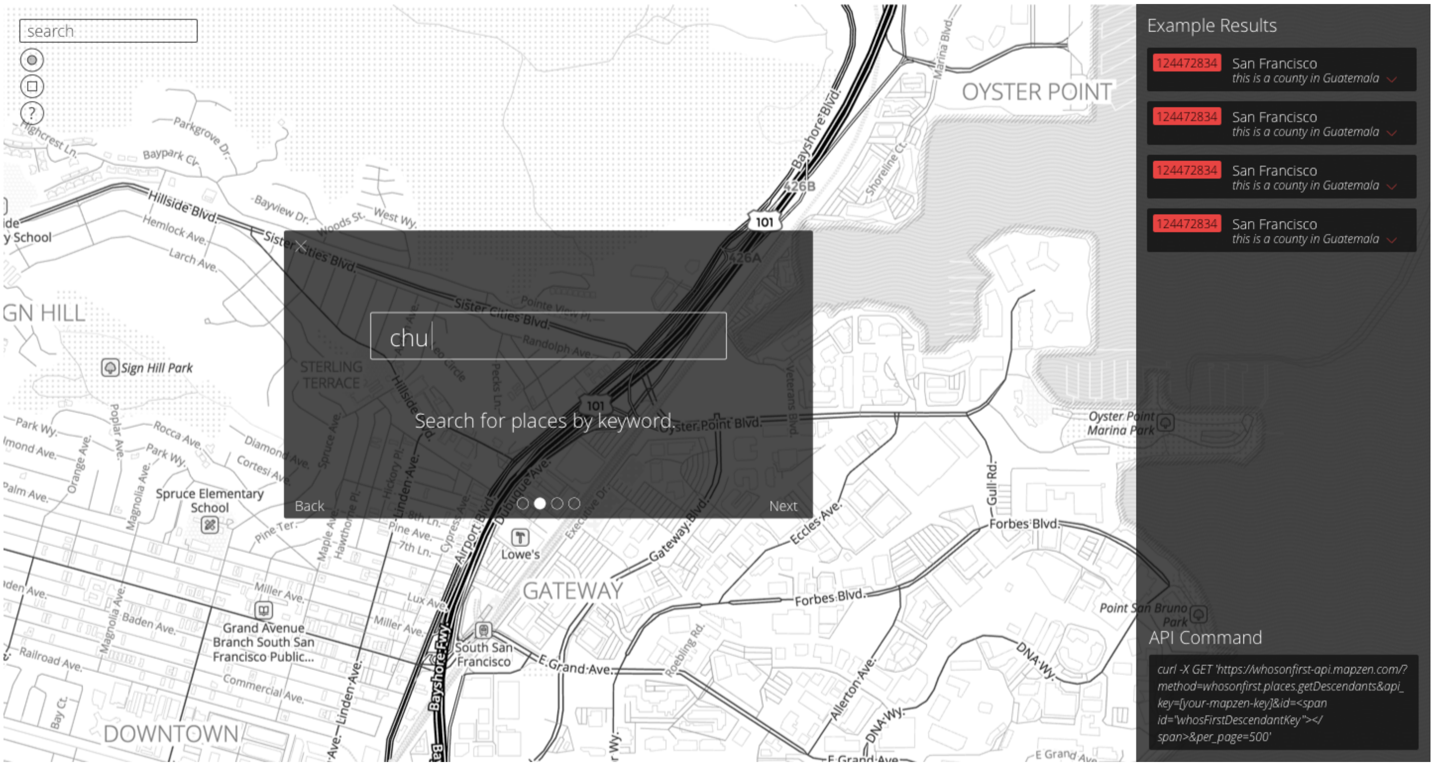
Click on the box icon, drag out a box or enter a minimum and maximum latitude and longtitude.

Latitude
or
Longtitude
Latitude
or
Longtitude
and
Range
meters
or
Minimum Latitude
Maximum Latitude
Minimum Longtitude
Maximum Longtitude

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/MapInteraction33Small.png)

After showing my team this set of mockups, they mentioned including descendants. I felt it would difficult for users to utilize the point and box functionality to find specific places and their descendants.

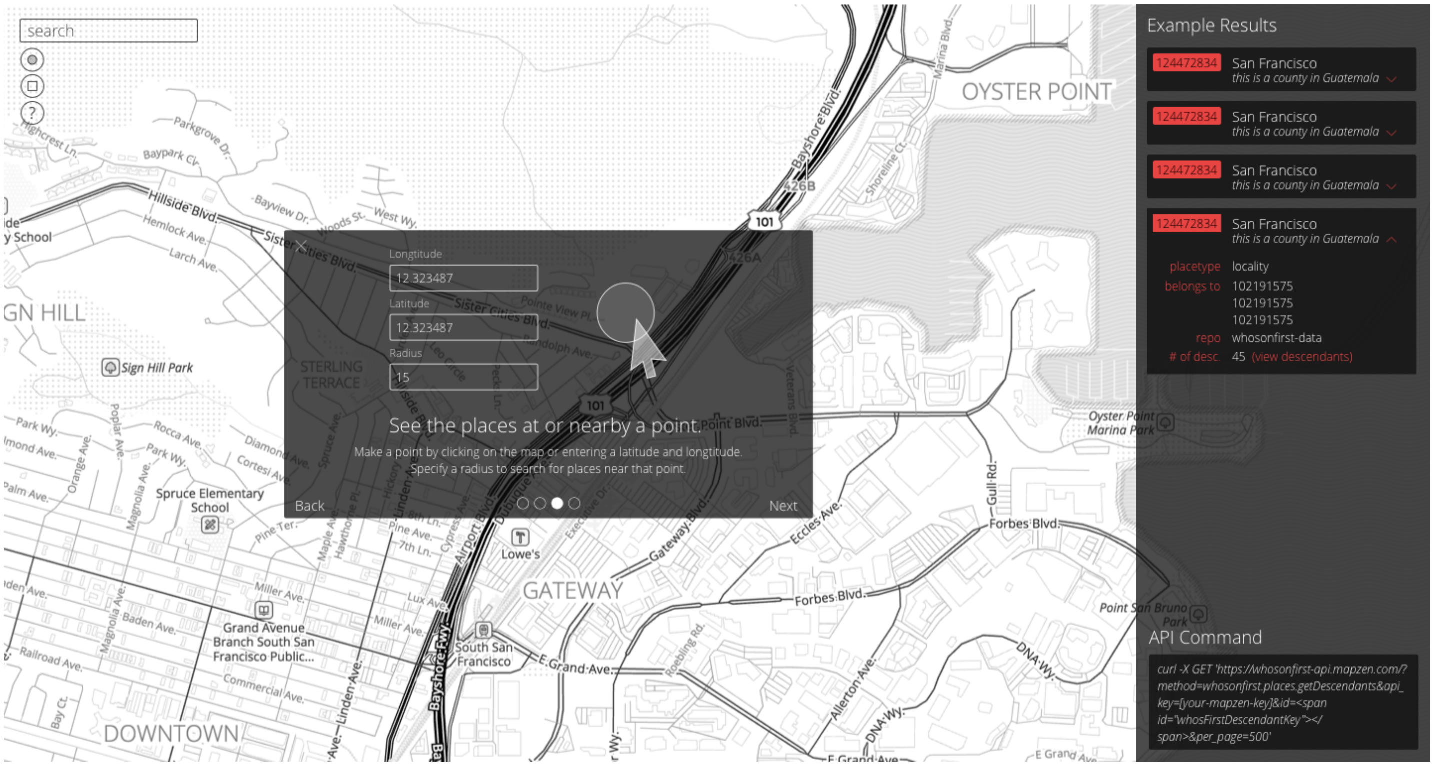
Users could search for a place and then view the descendants of those places, but the walkthrough would have to be broken up.


search
?
Example Results
124472834 San Francisco
this is a county in Guatemala
124472834 San Francisco
this is a county in Guatemala
124472834 San Francisco
this is a county in Guatemala
124472834 San Francisco
this is a county in Guatemala
chu
Back
Next
OYSTER POINT
GATEWAY
DOWNTOWN
Sign Hill Park
Spruce Elementary School
Grand Avenue Branch South San Francisco Public...
South San Francisco
Lowe's
Oyster Point Marina Park
Point San Bruno Park
Hillside Blvd.
Sister Cities Blvd.
Oyster Point Blvd.
Forbes Blvd.
Gull Rd.
Eccles Ave.
Gateway Blvd.
E Grand Ave.
DNA Wy.
Allerton Ave.
Bayshore Blvd.
101
API Command

Search for places by keyword.

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.getDescendants&api_key=[your-mapzen-key]&id=<span id="whosFirstDescendantKey"></span>&per_page=500'
```

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/MapInteraction41.png)


search
OYSTER POINT
GN HILL
STERLING TERRACE
GATEWAY
DOWNTOWN
Sign Hill Park
Spruce Elementary School
Lowe's
Grand Avenue Branch South San Francisco Public...
South San Francisco
Oyster Point Marina Park
Point San Bruno Park
Back
Next

Longtitude
12.323487
Latitude
12.323487
Radius
15

See the places at or nearby a point.

Make a point by clicking on the map or entering a latitude and longtitude.
Specify a radius to search for places near that point.

Example Results

124472834 San Francisco
this is a county in Guatemala

124472834 San Francisco
this is a county in Guatemala

124472834 San Francisco
this is a county in Guatemala

124472834 San Francisco
this is a county in Guatemala

placetype	locality
belongs to	102191575 102191575 102191575
repo	whosonfirst-data
# of desc.	45 (view descendants)

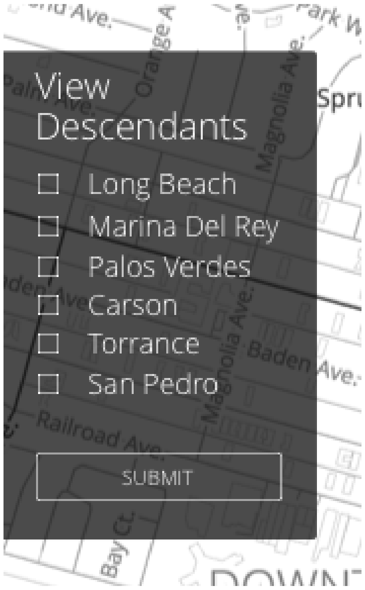
API Command

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?
method=whosonfirst.places.getDescendants&api_
key=[your-mapzen-key]&id=<span
id="whosFirstDescendantKey"></
span>&per_page=500'
```

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/MapInteraction46.png)

I also explored different possible implementations.

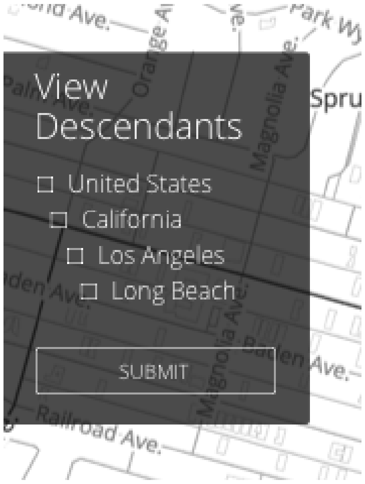
For instance, could you recognize a place, find that place's locality, find that locality's parent, and then use that parent to find different places on the same level as the original place's locality? You could then give users the ability to view the descendants of those localities.



View Descendants

- ☐ Long Beach
- ☐ Marina Del Rey
- ☐ Palos Verdes
- ☐ Carson
- ☐ Torrance
- ☐ San Pedro

Or could you provide users an interface to move from one level to another.



I ultimately believe that these implementations would not work as well as a search and would lead to more difficulty for the user.

Stats

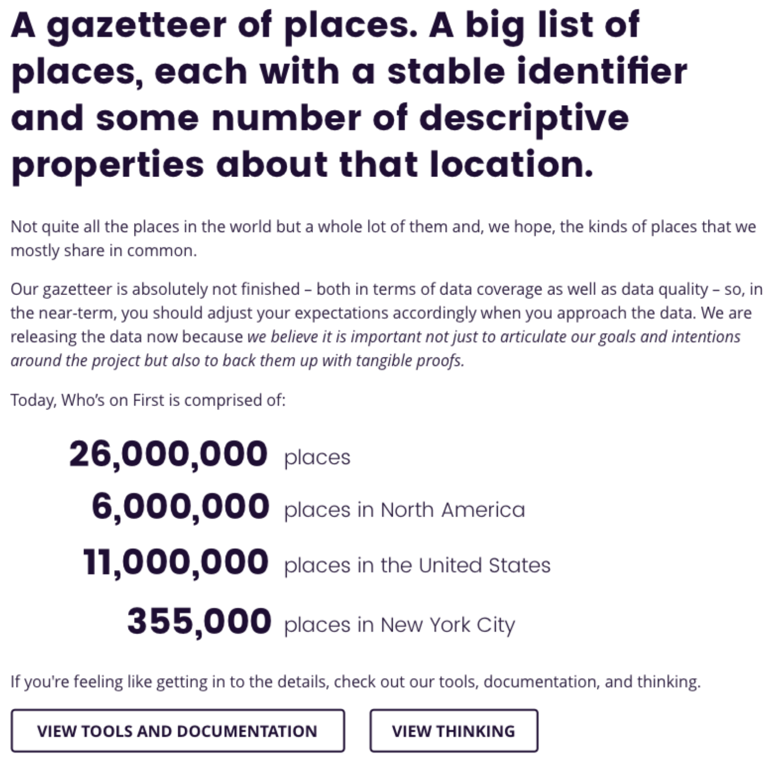
An additional way to create a dialog that could capture the attention of a user, while showing the depth of the project, would be through stats. WOF provides an ample amount of stats that could be relevant to a wide variety of users. At the same time, stats must be limited. You do not want to be consistently processing statistics in the background.

With the stats, I also wanted to avoid the parts of WOF that require background knowledge. Users shouldn’t have to discern what things mean, and the stats should be relatively simple.

So what statistics would users understand?

- *Total # of records - users definitely understand records when given some context*
- *Total # of countries - users understand countries too*
- *Total # of marine areas - a user may not immediately connect oceans, rivers, and seas with marines areas, maybe this one should be avoided*
- *Total # of regions - users have a general sense of regions*
- *Total # of counties - counties should be avoided, there are a greater number of localities than counties*
- *Total # of localities - users will not be able to immediately identify what a locality is, but we can add a small explanation that a locality is a city or town*
- *Total # of neighborhoods - probably does not fit, not every venue has a neighborhood, users from different areas may identify a neighborhood differently*
- *Total # of venues - users should understand this and it is at the core of WOF*

Further exploration needs to take place to understand how to organize and visualize the stats. Should it go from higher levels place types (countries, regions, etc…) to lower level place types (venues, etc…)? Should these stats be context specific? How much more computation would that require? Is that computation worth it?



A gazetteer of places. A big list of places, each with a stable identifier and some number of descriptive properties about that location.

Not quite all the places in the world but a whole lot of them and, we hope, the kinds of places that we mostly share in common.

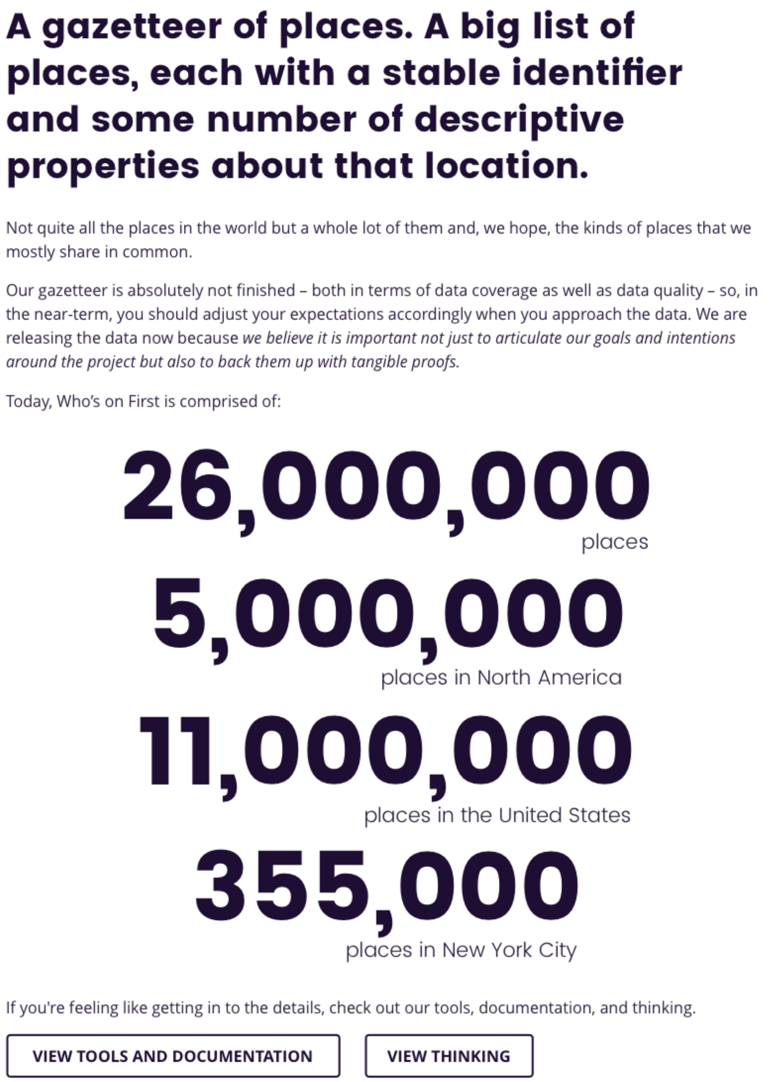
Our gazetteer is absolutely not finished – both in terms of data coverage as well as data quality – so, in the near-term, you should adjust your expectations accordingly when you approach the data. We are releasing the data now because *we believe it is important not just to articulate our goals and intentions around the project but also to back them up with tangible proofs.*

Today, Who's on First is comprised of:

26,000,000	places
6,000,000	places in North America
11,000,000	places in the United States
355,000	places in New York City

If you're feeling like getting in to the details, check out our tools, documentation, and thinking.

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock4Small.png)



A gazetteer of places. A big list of places, each with a stable identifier and some number of descriptive properties about that location.

Not quite all the places in the world but a whole lot of them and, we hope, the kinds of places that we mostly share in common.

Our gazetteer is absolutely not finished – both in terms of data coverage as well as data quality – so, in the near-term, you should adjust your expectations accordingly when you approach the data. We are releasing the data now because *we believe it is important not just to articulate our goals and intentions around the project but also to back them up with tangible proofs.*

Today, Who's on First is comprised of:

26,000,000
places

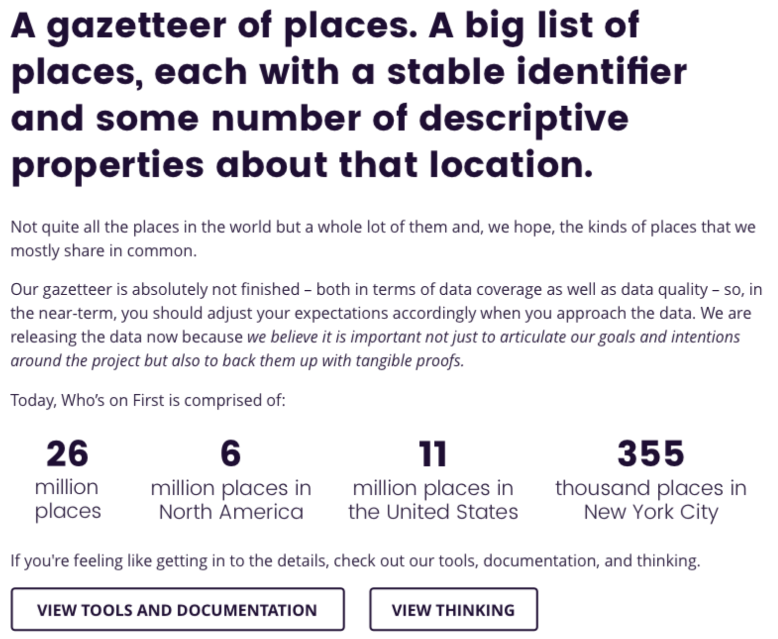
5,000,000
places in North America

11,000,000
places in the United States

355,000
places in New York City

If you're feeling like getting in to the details, check out our tools, documentation, and thinking.

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock5Small.png)



A gazetteer of places. A big list of places, each with a stable identifier and some number of descriptive properties about that location.

Not quite all the places in the world but a whole lot of them and, we hope, the kinds of places that we mostly share in common.

Our gazetteer is absolutely not finished – both in terms of data coverage as well as data quality – so, in the near-term, you should adjust your expectations accordingly when you approach the data. We are releasing the data now because *we believe it is important not just to articulate our goals and intentions around the project but also to back them up with tangible proofs.*

Today, Who's on First is comprised of:

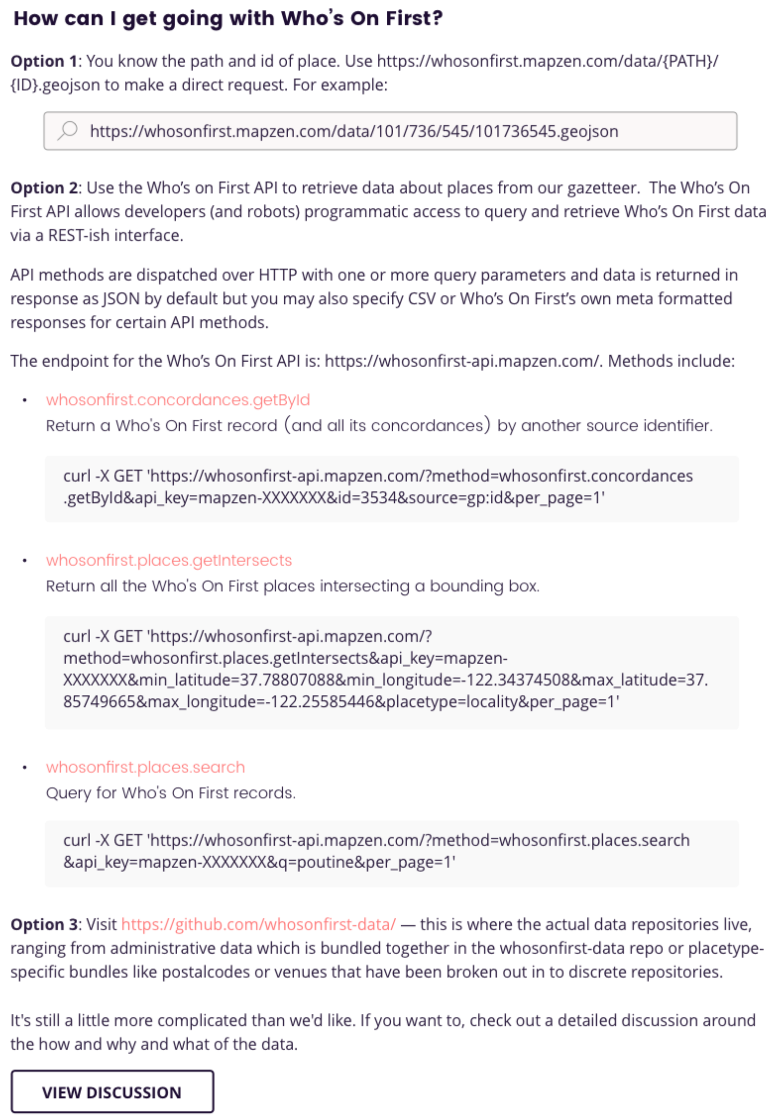
26	6	11	355
million places	million places in North America	million places in the United States	thousand places in New York City

If you're feeling like getting in to the details, check out our tools, documentation, and thinking.

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock6Small.png)

Getting Users Started

The website should also provide users with tools to immediately get started with WOF. This includes the different options a user could take to use WOF. For example, if a user knows the specific path and ID for a particular place, the website should provide that user with an example URL they could use to retrieve data. Or if the user wants to use the **WOF API (https://mapzen.com/documentation/wof/)**, that user should see a number of example queries and links to additional documentation. A link to **the Github repo where all WOF data lives (https://github.com/whosonfirst-data)** would also work, users could then download their own version of the data if they so desire. You can find a mockup below to see just how this could work.



How can I get going with Who's On First?

Option 1: You know the path and id of place. Use https://whosonfirst.mapzen.com/data/{PATH}/{ID}.geojson to make a direct request. For example:

https://whosonfirst.mapzen.com/data/101/736/545/101736545.geojson

Option 2: Use the Who's on First API to retrieve data about places from our gazetteer. The Who's On First API allows developers (and robots) programmatic access to query and retrieve Who's On First data via a REST-ish interface.

API methods are dispatched over HTTP with one or more query parameters and data is returned in response as JSON by default but you may also specify CSV or Who's On First's own meta formatted responses for certain API methods.

The endpoint for the Who's On First API is: https://whosonfirst-api.mapzen.com/. Methods include:

- whosonfirst.concordances.getById
  Return a Who's On First record (and all its concordances) by another source identifier.

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.concordances
.getById&api_key=mapzen-XXXXXXX&id=3534&source=gp:id&per_page=1'
```

- whosonfirst.places.getIntersects
  Return all the Who's On First places intersecting a bounding box.

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?
method=whosonfirst.places.getIntersects&api_key=mapzen-
XXXXXXX&min_latitude=37.78807088&min_longitude=-122.34374508&max_latitude=37.
85749665&max_longitude=-122.25585446&placetype=locality&per_page=1'
```

- whosonfirst.places.search
  Query for Who's On First records.

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.search
&api_key=mapzen-XXXXXXX&q=poutine&per_page=1'
```

Option 3: Visit https://github.com/whosonfirst-data/ — this is where the actual data repositories live, ranging from administrative data which is bundled together in the whosonfirst-data repo or placetype-specific bundles like postalcodes or venues that have been broken out in to discrete repositories.

It's still a little more complicated than we'd like. If you want to, check out a detailed discussion around the how and why and what of the data.

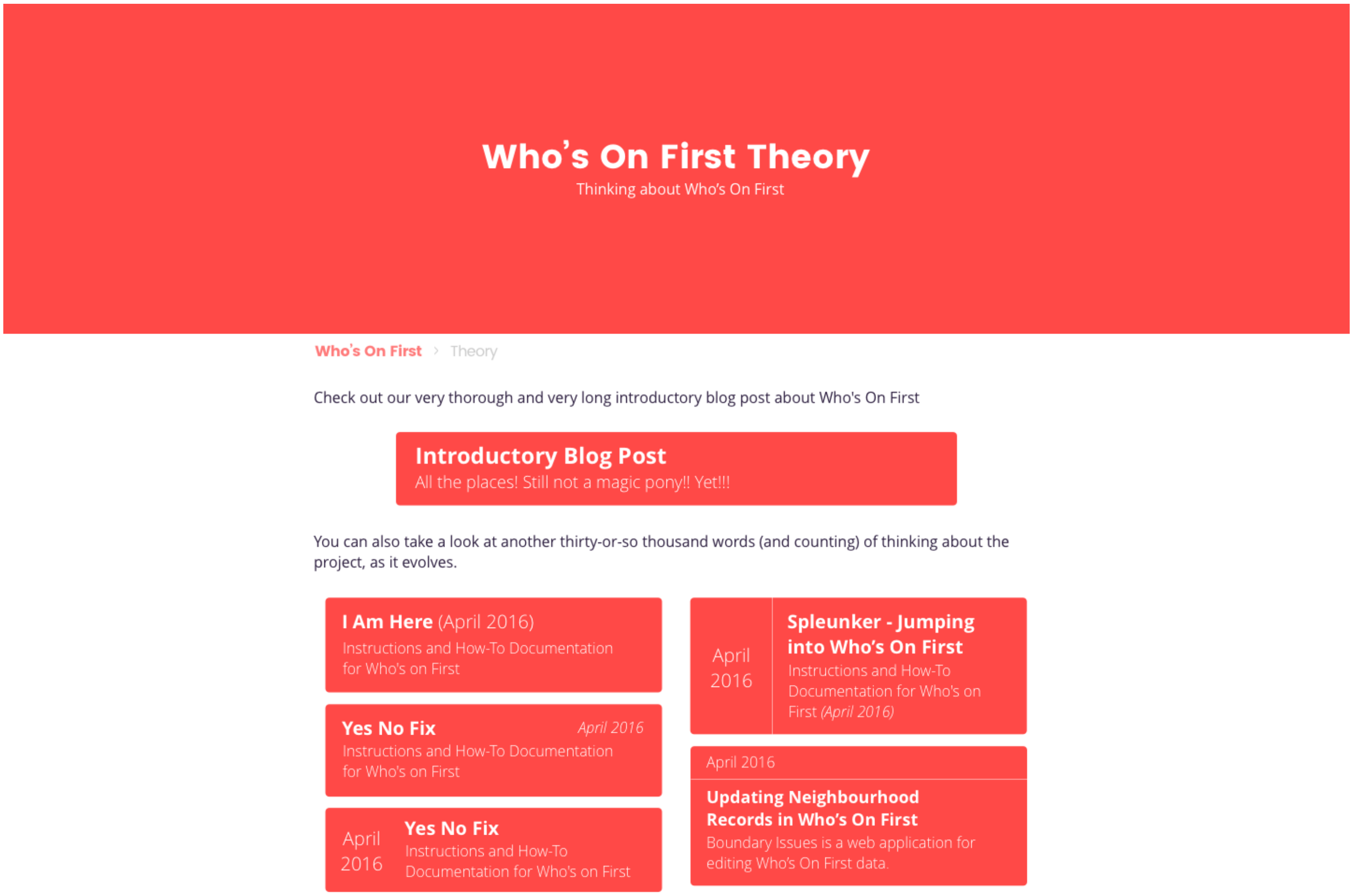
See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstMock12Small.png)

Information Architecture

Currently, part of WOF lives solely on **Github (https://github.com/whosonfirst,)**. One example of this is **source information (https://github.com/whosonfirst/whosonfirst-sources)**. In order to make WOF more useable we plan to clone selected parts of the information on Github to the website.

Some subpages already exist, for example the theory and data page, but at the very least there should be a page for tools, the blog, data, and docs. All these topics are complicated. By creating separate pages, less emphasis can be paid to those subjects on the home page.

We are currently working on just how all this information will be organized. Below you can find some mockups.



Who's On First Theory

Thinking about Who's On First

Who's On First › Theory

Check out our very thorough and very long introductory blog post about Who's On First

Introductory Blog Post
All the places! Still not a magic pony!! Yet!!!

You can also take a look at another thirty-or-so thousand words (and counting) of thinking about the project, as it evolves.

I Am Here (April 2016)
Instructions and How-To Documentation for Who's on First

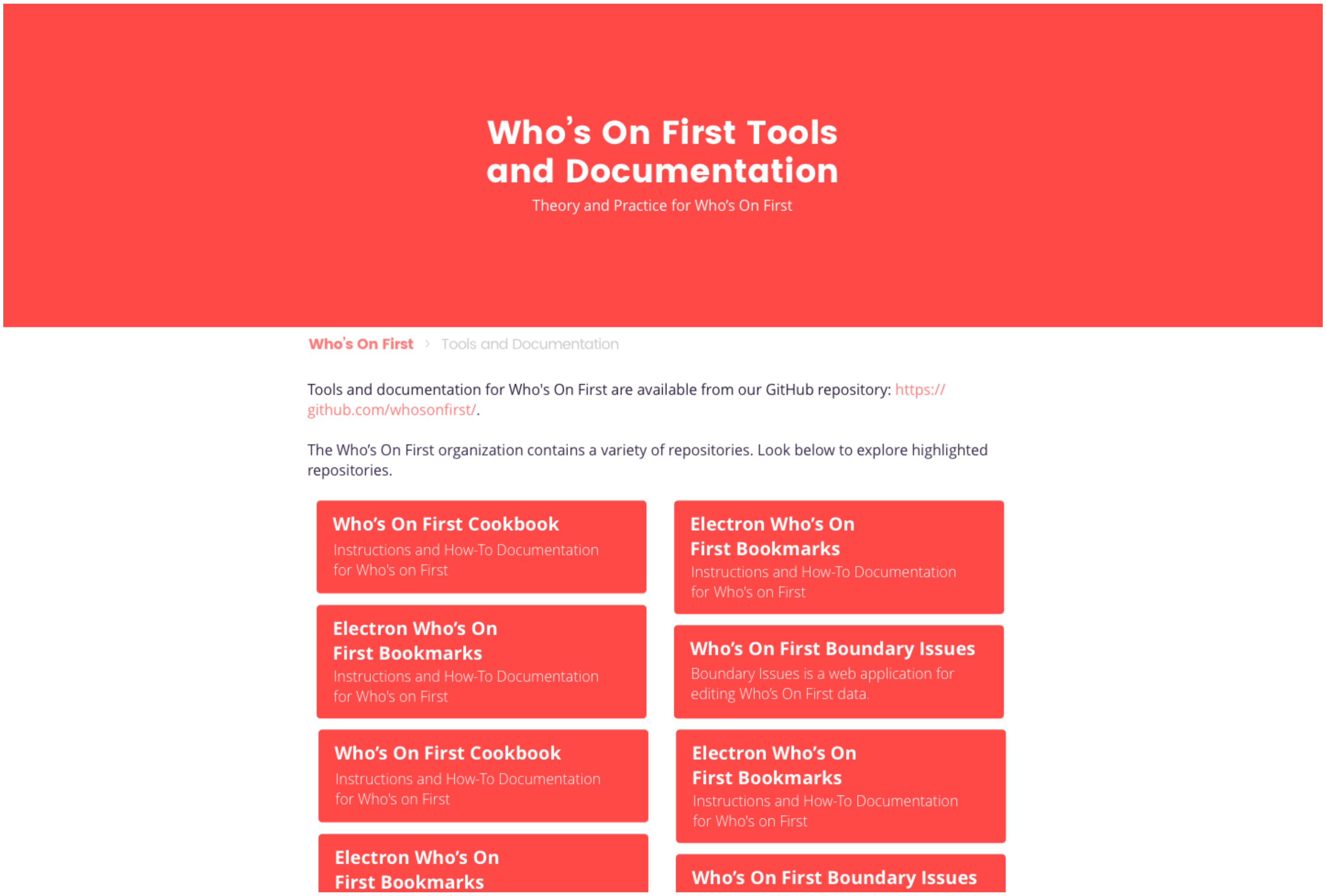
Yes No Fix *April 2016*
Instructions and How-To Documentation for Who's on First

April 2016 **Yes No Fix**
Instructions and How-To Documentation for Who's on First

April 2016 **Spleunker - Jumping into Who's On First**
Instructions and How-To Documentation for Who's on First *(April 2016)*

April 2016
Updating Neighbourhood Records in Who's On First
Boundary Issues is a web application for editing Who's On First data.

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstSubpageMock4.png)



Who’s On First Tools and Documentation

Theory and Practice for Who’s On First

Who’s On First › Tools and Documentation

Tools and documentation for Who's On First are available from our GitHub repository: https://github.com/whosonfirst/.

The Who’s On First organization contains a variety of repositories. Look below to explore highlighted repositories.

Who’s On First Cookbook
Instructions and How-To Documentation for Who's on First

Electron Who’s On First Bookmarks
Instructions and How-To Documentation for Who's on First

Electron Who’s On First Bookmarks
Instructions and How-To Documentation for Who's on First

Who’s On First Boundary Issues
Boundary Issues is a web application for editing Who’s On First data.

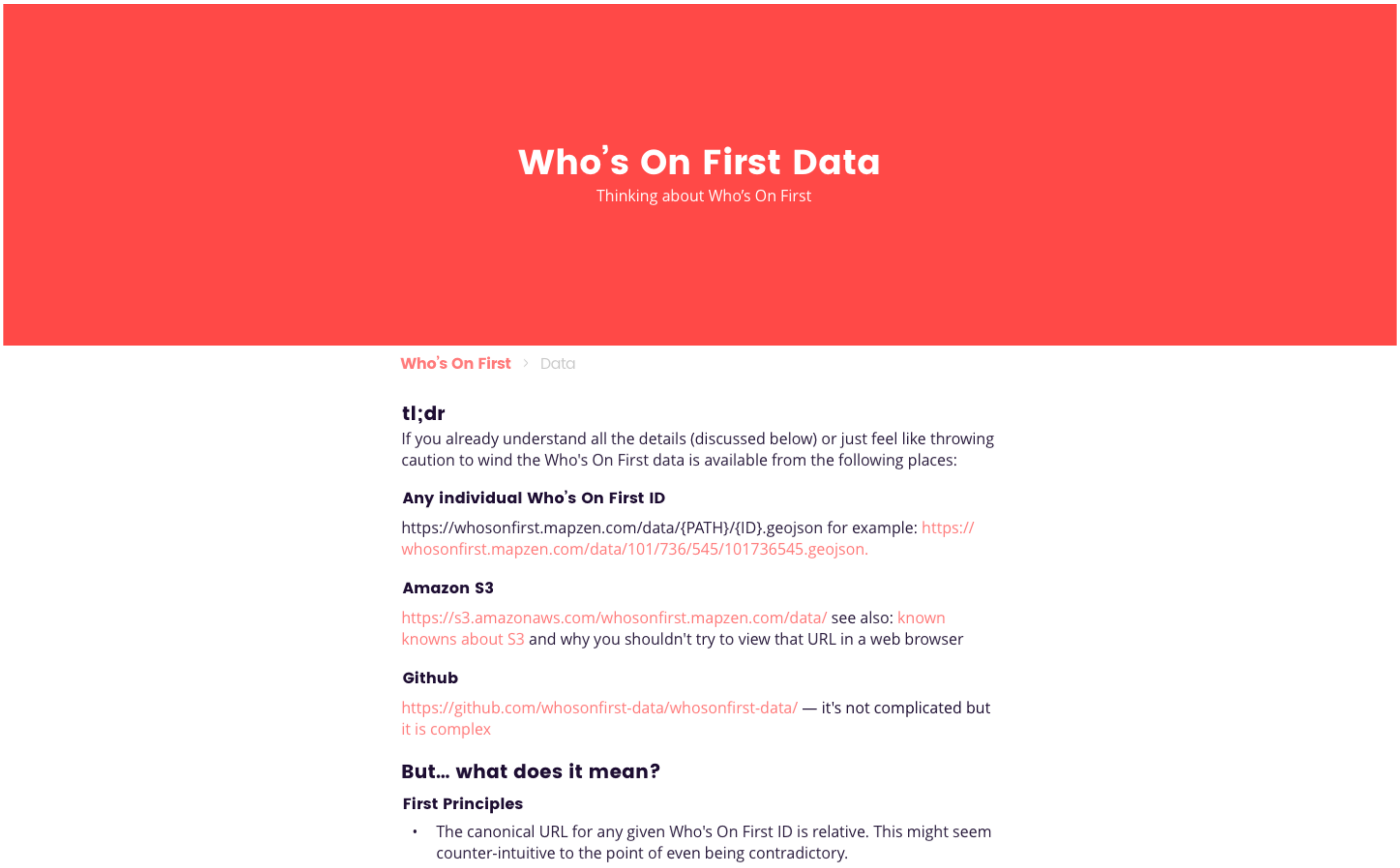
Who’s On First Cookbook
Instructions and How-To Documentation for Who's on First

Electron Who’s On First Bookmarks
Instructions and How-To Documentation for Who's on First

Electron Who’s On First Bookmarks

Who’s On First Boundary Issues

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstSubpageMock2.png)



Who’s On First Data

Thinking about Who’s On First

Who’s On First › Data

tl;dr

If you already understand all the details (discussed below) or just feel like throwing caution to wind the Who's On First data is available from the following places:

Any individual Who’s On First ID

https://whosonfirst.mapzen.com/data/{PATH}/{ID}.geojson for example: https://whosonfirst.mapzen.com/data/101/736/545/101736545.geojson.

Amazon S3

https://s3.amazonaws.com/whosonfirst.mapzen.com/data/ see also: known knowns about S3 and why you shouldn't try to view that URL in a web browser

Github

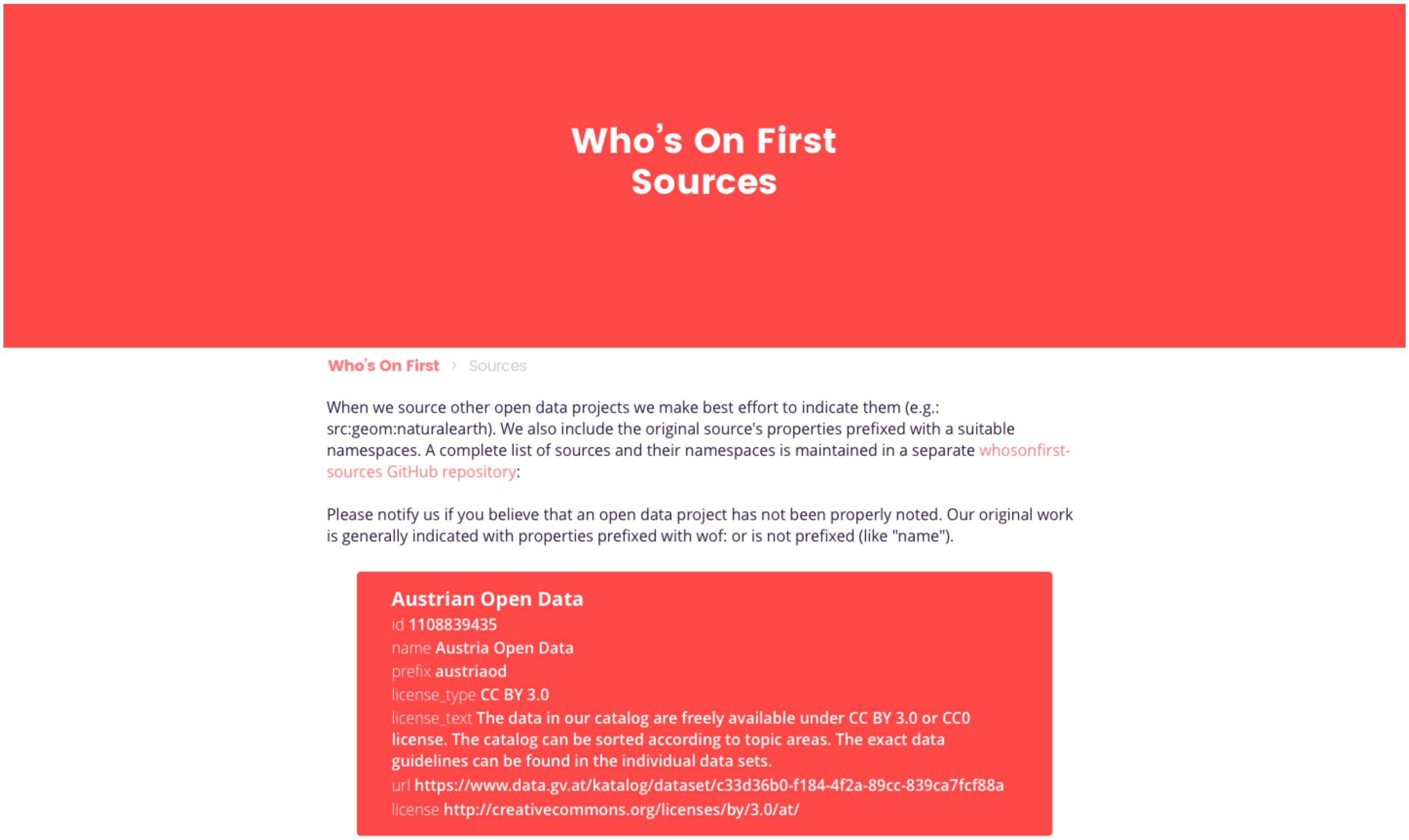
https://github.com/whosonfirst-data/whosonfirst-data/ — it's not complicated but it is complex

But... what does it mean?

First Principles

- The canonical URL for any given Who's On First ID is relative. This might seem counter-intuitive to the point of even being contradictory.

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstSubpageMock6.png)



Who's On First Sources

Who's On First › Sources

When we source other open data projects we make best effort to indicate them (e.g.: src:geom:naturalearth). We also include the original source's properties prefixed with a suitable namespaces. A complete list of sources and their namespaces is maintained in a separate whosonfirst-sources GitHub repository:

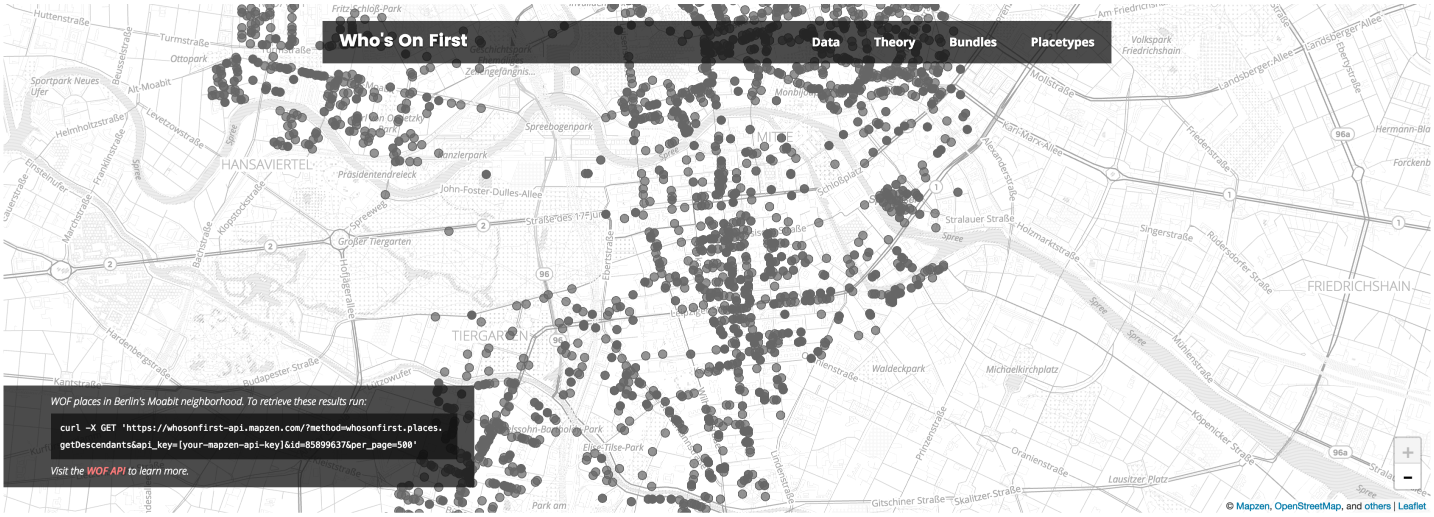
Please notify us if you believe that an open data project has not been properly noted. Our original work is generally indicated with properties prefixed with wof: or is not prefixed (like "name").

Austrian Open Data
id **1108839435**
name **Austria Open Data**
prefix **austriaod**
license_type **CC BY 3.0**
license_text **The data in our catalog are freely available under CC BY 3.0 or CC0 license. The catalog can be sorted according to topic areas. The exact data guidelines can be found in the individual data sets.**
url **https://www.data.gv.at/katalog/dataset/c33d36b0-f184-4f2a-89cc-839ca7fcf88a**
license **http://creativecommons.org/licenses/by/3.0/at/**

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/WhosOnFirstSubpageMock8.png)

Where Are We Now?

Below you will see how the current site looks in a development environment. We just pushed this out to production.

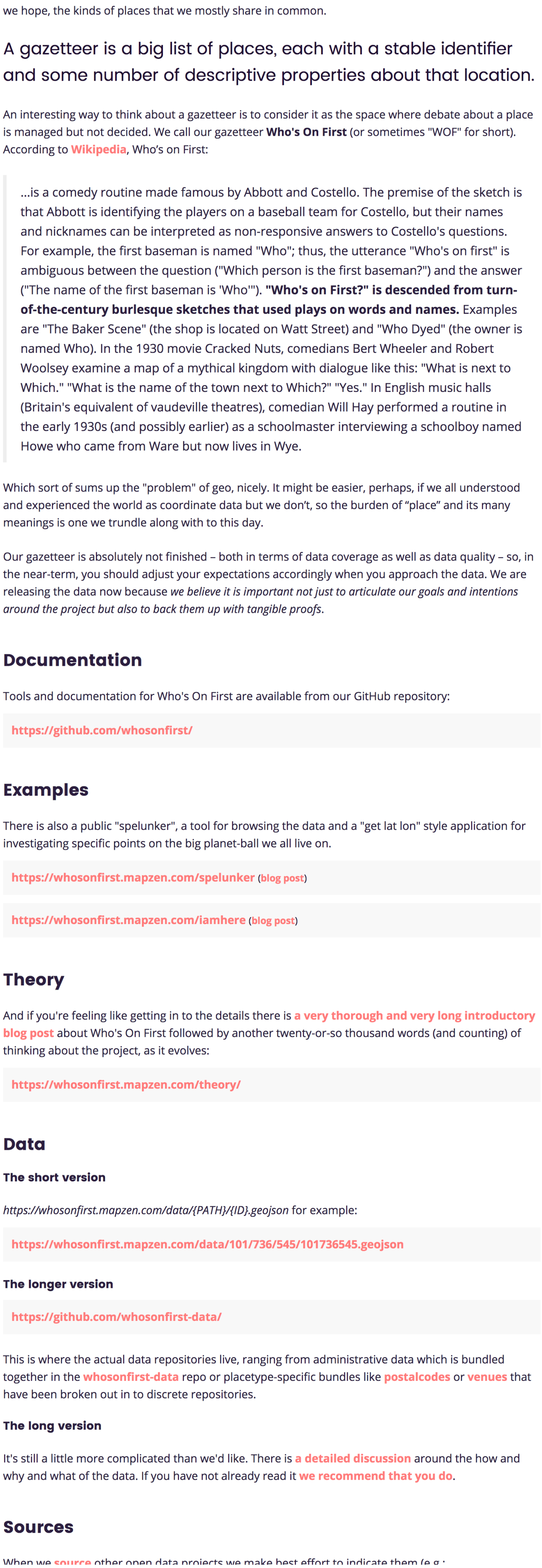

Who's On First
Data
Theory
Bundles
Placetypes
HANSAVIERTEL
TIERGARTEN
MITTE
FRIEDRICHSHAIN
Großer Tiergarten
Spreebogenpark
Kantstraße
Hardenbergstraße
Budapester Straße
Lützowufer
Kleiststraße
Karl-Marx-Allee
Holzmarktstraße
Stralauer Straße
Waldeckpark
Michaelkirchplatz
Oranienstraße
Lausitzer Platz
+
-
© Mapzen, OpenStreetMap, and others | Leaflet

WOF places in Berlin's Moabit neighborhood. To retrieve these results run:

```
curl -X GET 'https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.
getDescendants&api_key=[your-mapzen-api-key]&id=85899637&per_page=500'
```

Visit the WOF API to learn more.

Who's On First: "better than yesterday"



Who's On First: "better than yesterday"

Who's On First is a gazetteer of places. Not quite *all* the places in the world but a whole lot of them and, we hope, the kinds of places that we mostly share in common.

A gazetteer is a big list of places, each with a stable identifier and some number of descriptive properties about that location.

An interesting way to think about a gazetteer is to consider it as the space where debate about a place is managed but not decided. We call our gazetteer **Who's On First** (or sometimes "WOF" for short). According to **Wikipedia**, Who's on First:

> ...is a comedy routine made famous by Abbott and Costello. The premise of the sketch is that Abbott is identifying the players on a baseball team for Costello, but their names and nicknames can be interpreted as non-responsive answers to Costello's questions. For example, the first baseman is named "Who"; thus, the utterance "Who's on first" is ambiguous between the question ("Which person is the first baseman?") and the answer ("The name of the first baseman is 'Who'"). **"Who's on First?" is descended from turn-of-the-century burlesque sketches that used plays on words and names.** Examples are "The Baker Scene" (the shop is located on Watt Street) and "Who Dyed" (the owner is named Who). In the 1930 movie Cracked Nuts, comedians Bert Wheeler and Robert Woolsey examine a map of a mythical kingdom with dialogue like this: "What is next to Which." "What is the name of the town next to Which?" "Yes." In English music halls (Britain's equivalent of vaudeville theatres), comedian Will Hay performed a routine in the early 1930s (and possibly earlier) as a schoolmaster interviewing a schoolboy named Howe who came from Ware but now lives in Wye.

Which sort of sums up the "problem" of geo, nicely. It might be easier, perhaps, if we all understood and experienced the world as coordinate data but we don't, so the burden of "place" and its many meanings is one we trundle along with to this day.

Our gazetteer is absolutely not finished – both in terms of data coverage as well as data quality – so, in the near-term, you should adjust your expectations accordingly when you approach the data. We are releasing the data now because *we believe it is important not just to articulate our goals and intentions around the project but also to back them up with tangible proofs*.

Documentation

Tools and documentation for Who's On First are available from our GitHub repository:

https://github.com/whosonfirst/

Examples

There is also a public "spelunker", a tool for browsing the data and a "get lat lon" style application for investigating specific points on the big planet-ball we all live on.

https://whosonfirst.mapzen.com/spelunker (**blog post**)

https://whosonfirst.mapzen.com/iamhere (**blog post**)

Theory

And if you're feeling like getting in to the details there is **a very thorough and very long introductory blog post** about Who's On First followed by another twenty-or-so thousand words (and counting) of thinking about the project, as it evolves:

https://whosonfirst.mapzen.com/theory/

Data

The short version

https://whosonfirst.mapzen.com/data/{PATH}/{ID}.geojson for example:

https://whosonfirst.mapzen.com/data/101/736/545/101736545.geojson

The longer version

https://github.com/whosonfirst-data/

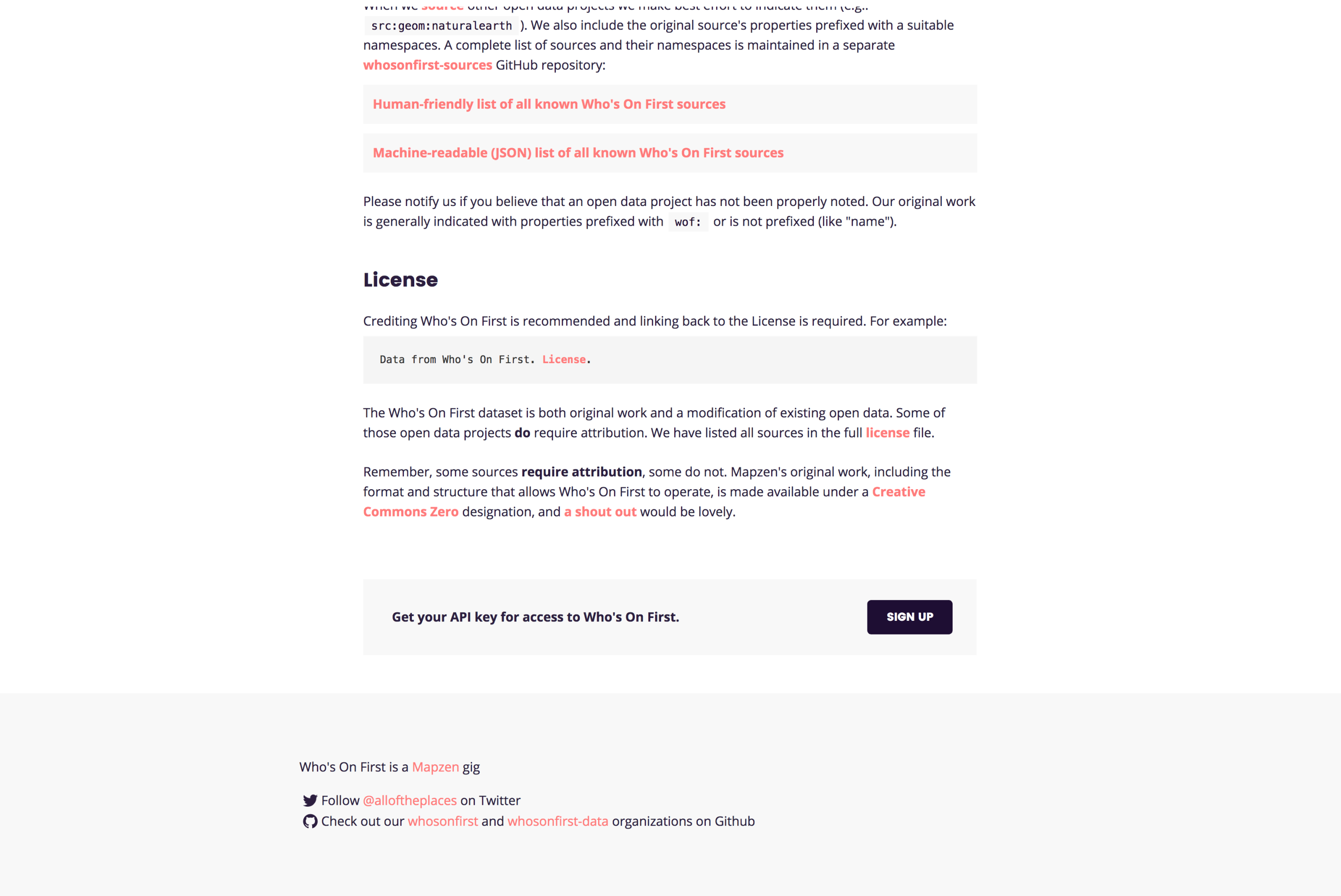
This is where the actual data repositories live, ranging from administrative data which is bundled together in the **whosonfirst-data** repo or placetype-specific bundles like **postalcodes** or **venues** that have been broken out in to discrete repositories.

The long version

It's still a little more complicated than we'd like. There is **a detailed discussion** around the how and why and what of the data. If you have not already read it **we recommend that you do**.

Sources

When we **source** other open data projects we make best effort to indicate them (e.g.:



When we source other open data projects we make best effort to indicate them (e.g. `src:geom:naturalearth`). We also include the original source's properties prefixed with a suitable namespaces. A complete list of sources and their namespaces is maintained in a separate whosonfirst-sources GitHub repository:

Human-friendly list of all known Who's On First sources

Machine-readable (JSON) list of all known Who's On First sources

Please notify us if you believe that an open data project has not been properly noted. Our original work is generally indicated with properties prefixed with `wof:` or is not prefixed (like "name").

License

Crediting Who's On First is recommended and linking back to the License is required. For example:

```
Data from Who's On First. License.
```

The Who's On First dataset is both original work and a modification of existing open data. Some of those open data projects **do** require attribution. We have listed all sources in the full license file.

Remember, some sources **require attribution**, some do not. Mapzen's original work, including the format and structure that allows Who's On First to operate, is made available under a **Creative Commons Zero** designation, and **a shout out** would be lovely.

Get your API key for access to Who's On First.

Who's On First is a Mapzen gig

Follow @alloftheplaces on Twitter
Check out our whosonfirst and whosonfirst-data organizations on Github

See full-size image (https://s3.amazonaws.com/mapzen-assets/images/wof-website-resdesign/whoswholescreen.png)

Where Are We Going To Be?

Over the course of the next couple weeks, we plan to make further improvements to the website in the following areas:

- Building out the interactive component.
- Creating a narrative around WOF stats.
- Working on the information architecture of the website.
- Deciding what information lives on the website and what information lives on Github.

Let us know what you think at @**alloftheplaces (https://twitter.com/alloftheplaces)**!

· 28 July 2017 ·

Scott Dombkowski

Scott is a summer 2017 intern on the Mapzen data team, a grad student at Carnegie Mellon's School of Design, and a rearrangeable furniture buff.

© 2017 Mapzen

Geotagging WOF venues

boundaryissues (/tag/boundaryissues) whosonfirst (/tag/whosonfirst) data (/tag/data)

For much of this summer I've been working remotely on Who's On First. In early July I was in Korea giving **a workshop on offline mapping (http://seoulbiennale.org/en/public-programs/educational/read-write-offline-mapping)** as part of the **Seoul Biennale of Architecture and Urbanism (http://seoulbiennale.org/en/)**. From there I joined my wife **Ellie (https://ellieirons.com/)** at an **artist residency in Taipei (http://bambooculture.com/en)**, where I've been working 12 hours ahead of my **EDT (https://whosonfirst.mapzen.com/spelunker/id/102047399/)** colleagues. Along the way I've been collecting new places for WOF, since both **Taiwan (https://github.com/whosonfirst-data/whosonfirst-data-venue-tw)** and **South Korea (https://github.com/whosonfirst-data/whosonfirst-data-venue-kr)** are still rather sparsely populated with venues. This was my excuse to try using my software "in the field" and expand WOF's venue coverage.

This post includes a fair share of obscure technical details. Perhaps it might be useful for someone working on a similar application. (And shout-out to the NYPL's **Bert Spaan (http://bertspaan.nl/)** who's released a very interesting looking **geotagging Leaflet plugin (https://github.com/nypl-spacetime/Leaflet.GeotagPhoto)**.)

Start where you are

I started with what I already had running on **Boundary Issues (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/)**, I just loaded it up on my phone. (Here is where I mention that the bespoke web-based editor I'm working on for Who's On First is not quite ready for public use, but **I'm blogging about it (https://mapzen.com/tag/boundaryissues)** out in the open anyway. **Drop us a line (mailto:boundaryissues@mapzen.com)** if you think you have something to add.) In addition to the general purpose **WOF properties editor (https://mapzen.com/blog/boundary-issues-properties/)**, I've been working on a pared down **venue-specific interface (https://mapzen.com/blog/simple-is-hard/)**. I used that simpler venues UI to try adding some new places I've enjoyed during my travels.

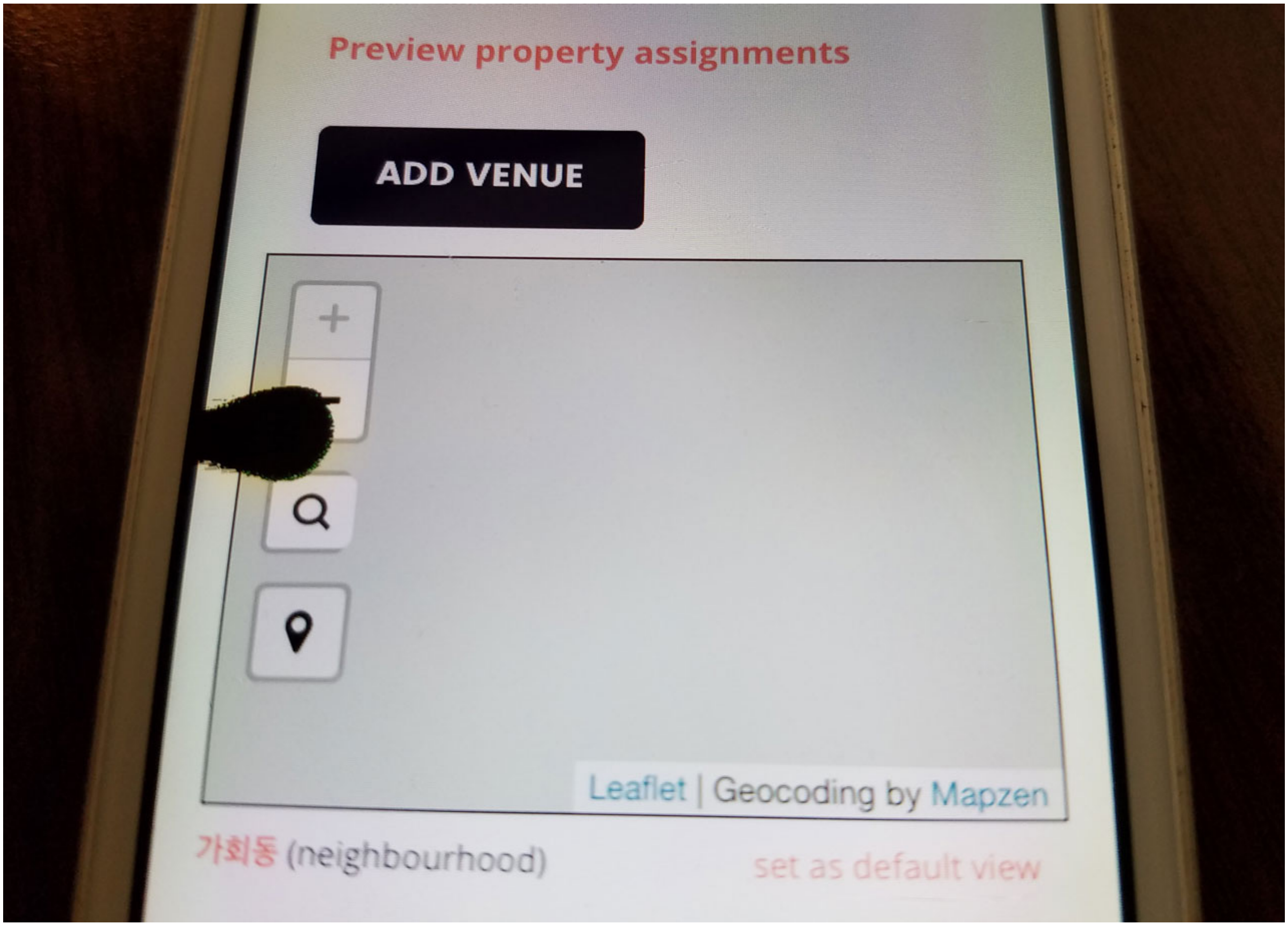
I am carrying two phones with me. One is an iPhone 5 with T-Mobile's generous data roaming—a throttled, but functional, version of the mobile data service I get at home (at no extra cost). The other is a Samsung Galaxy S7 with Verizon service, which works great, but not outside the USA. I've been sticking to wifi hotspots on the Android phone while I'm traveling.

Slow bandwidth is very real

Instead of investing in a Korean or Taiwanese SIM card, I decided to embrace my slower bandwidth as a useful constraint. By sticking to my throttled roaming data, I'm getting a bit of a taste for what mobile bandwidth bottlenecks feel like day-to-day. My iPhone is limited to 3G networks even though everyone around me has fast LTE connectivity.

I've gotten better at planning ahead, making a deliberate effort to cache the map data (and routing results) I know I'll need to navigate the city. In NYC, my network experience alternates between fast cable, fast mobile data (with a subway break for reading a book), and then back to a fast fiber connection. It's easy to forget about all the slow connectivity that's out here.

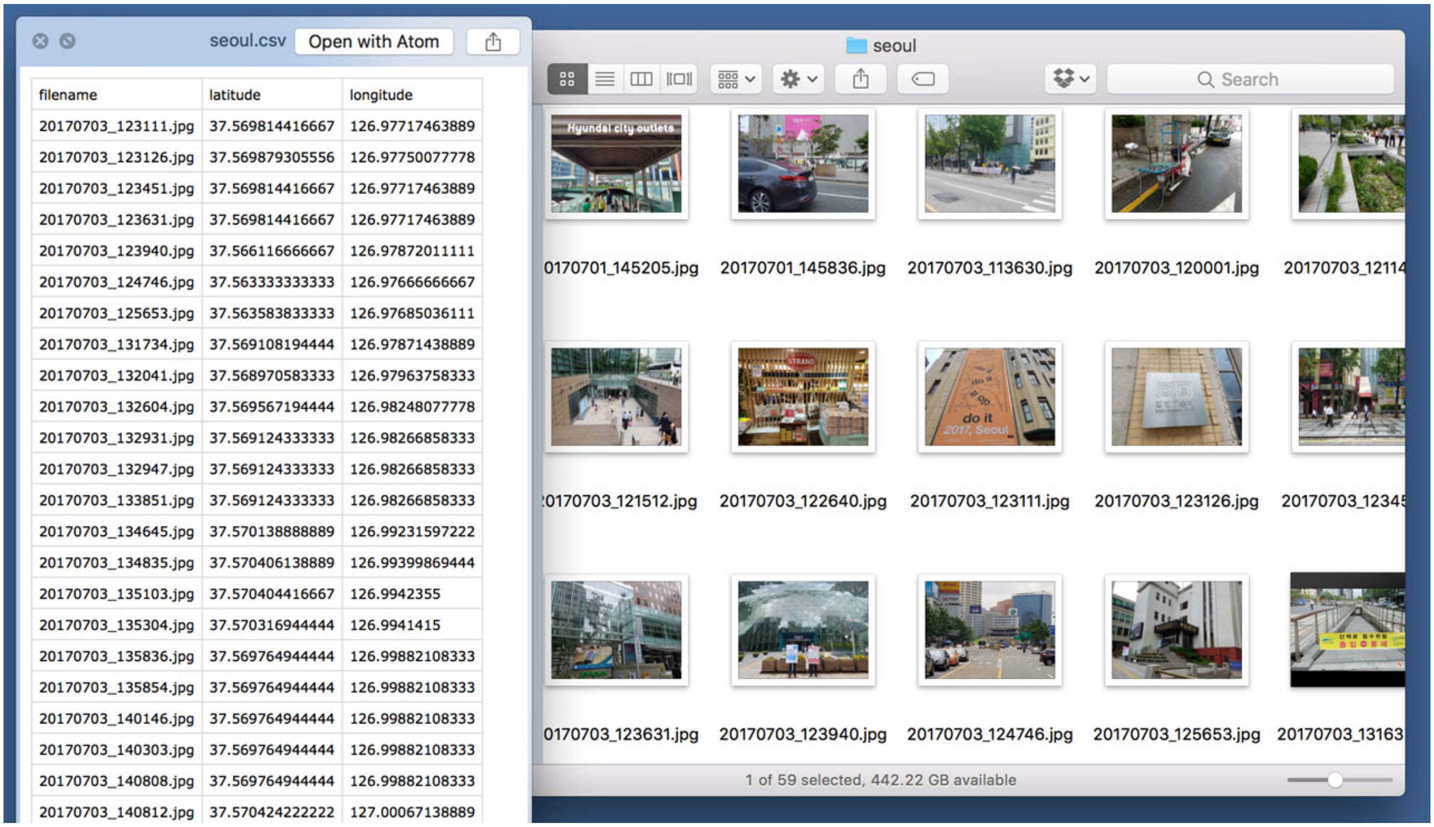
In testing out Boundary Issues, the first thing I learned is that mobile web slippy maps don't really work so well on a bandwidth-constrained network. I would load up the "Add A Venue" page and just stare at the place where a map should appear. After waiting long enough, the screen would turn off to save battery and I'd mutter something rude under my breath.



The “headless” option

I decided to try a different approach. What if instead of relying on a mobile web page, I used the latitude/longitude metadata embedded in the photos I took? I could just take pictures and then process them when I got back to a faster internet connection. This had the added benefit of making data collection more enjoyable. Instead of staring down at my pocket supercomputer all the time, I’d just take a quick photo from the lock screen shortcut, and it’d go right back into my pocket.

I took a bunch of photos around Seoul with each of my phones. This method was so much faster, and felt less battery intensive compared to my mobile web interface. Once I got back to my laptop, I downloaded the photos and ran them through a **command-line script (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/blob/master/bin/jpgs2csv.php)** to convert a folder of images into a CSV file containing a list of filenames and latitude/longitude coordinates.



seoul.csv
Open with Atom
seoul
Search
0170701_145205.jpg
20170701_145836.jpg
20170703_113630.jpg
20170703_120001.jpg
20170703_12114
0170703_121512.jpg
20170703_122640.jpg
20170703_123111.jpg
20170703_123126.jpg
20170703_12345
0170703_123631.jpg
20170703_123940.jpg
20170703_124746.jpg
20170703_125653.jpg
20170703_13163
1 of 59 selected, 442.22 GB available

filename	latitude	longitude
20170703_123111.jpg	37.569814416667	126.97717463889
20170703_123126.jpg	37.569879305556	126.97750077778
20170703_123451.jpg	37.569814416667	126.97717463889
20170703_123631.jpg	37.569814416667	126.97717463889
20170703_123940.jpg	37.566116666667	126.97872011111
20170703_124746.jpg	37.563333333333	126.97666666667
20170703_125653.jpg	37.563583833333	126.97685036111
20170703_131734.jpg	37.569108194444	126.97871438889
20170703_132041.jpg	37.568970583333	126.97963758333
20170703_132604.jpg	37.569567194444	126.98248077778
20170703_132931.jpg	37.569124333333	126.98266858333
20170703_132947.jpg	37.569124333333	126.98266858333
20170703_133851.jpg	37.569124333333	126.98266858333
20170703_134645.jpg	37.570138888889	126.99231597222
20170703_134835.jpg	37.570406138889	126.99399869444
20170703_135103.jpg	37.570404416667	126.9942355
20170703_135304.jpg	37.570316944444	126.9941415
20170703_135836.jpg	37.569764944444	126.99882108333
20170703_135854.jpg	37.569764944444	126.99882108333
20170703_140146.jpg	37.569764944444	126.99882108333
20170703_140303.jpg	37.569764944444	126.99882108333
20170703_140808.jpg	37.569764944444	126.99882108333
20170703_140812.jpg	37.570424222222	127.00067138889

I took that CSV file and imported it into Boundary Issues, using the **step-through venue importer (https://mapzen.com/blog/weird-and-wonderful/)**. And it worked! The one flaw in this setup was matching up each filename from the CSV with its corresponding photo in order to remember the place names.

At first I opted for my S7's higher quality camera. But then I noticed Android has a tendency to hold onto an outdated location compared to iOS. Now I prefer the lower-quality photos from my old iPhone, favoring its more consistently accurate geotags. (Perhaps there is some setting in Android to improve its accuracy that I'm missing.)

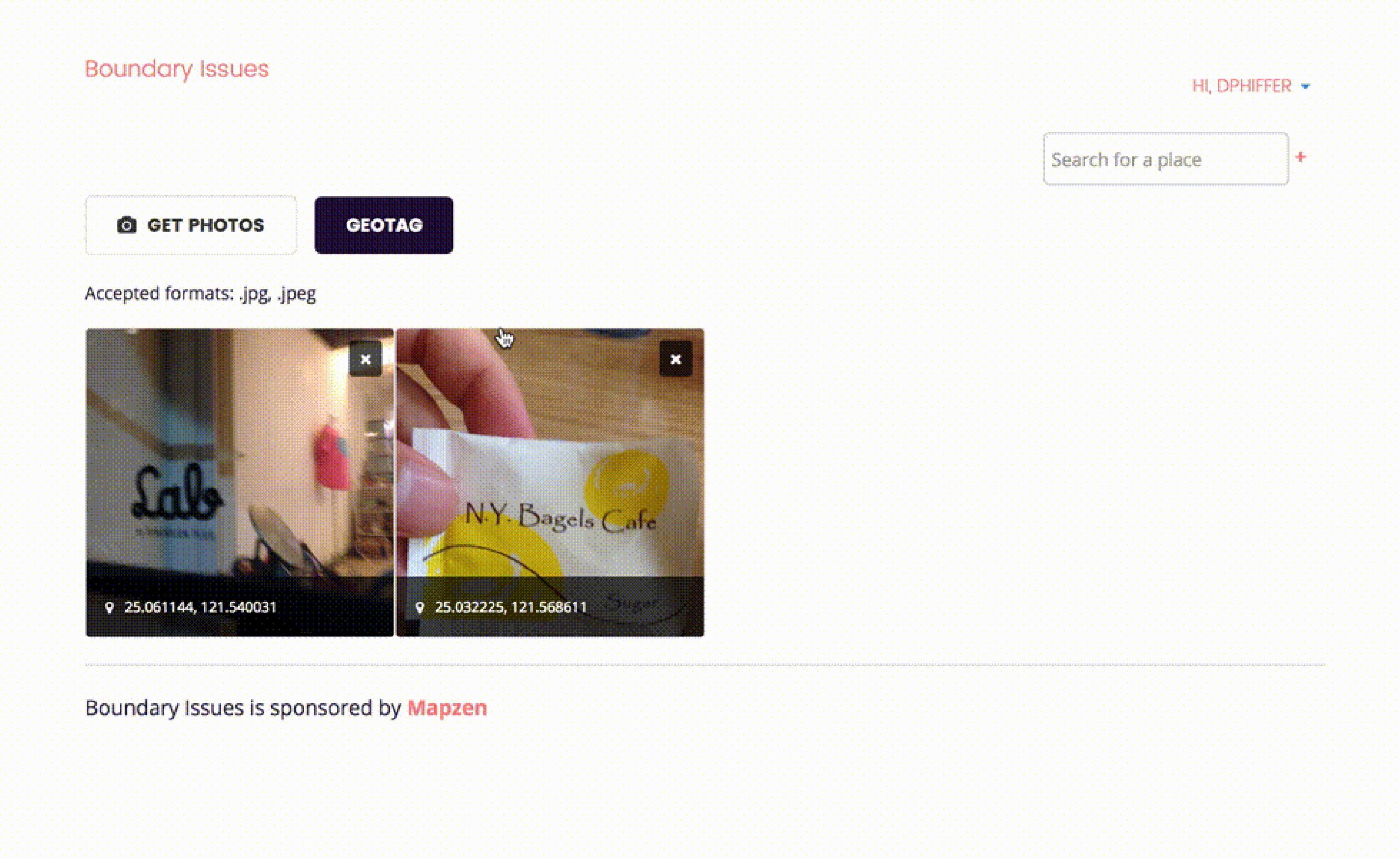
Skipping the CSV stage

After running a few CSV venue imports, I started to wonder if I could just put the visual cue I'd photographed *on the page* next to the extracted geotags. I continue to be astonished at the range of things JavaScript can do. As it turns out, you can **import/cache JPEG binary data (https://gist.github.com/straup/0ee19019afd88c6dbd2d)** and **extract EXIF tags (https://github.com/exif-js/exif-js)** without hitting any server-side code.

The newest incarnation of the geotagged importer is basically an **HTML file input (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/file)** wired up to some fancy JavaScript that eventually uploads a GeoJSON WOF record into Boundary Issues.

With some important caveats (discussed below), this approach lets you keep your geotagged photo on your own device(s). Boundary Issues inspects the geotags on the client-side, so it doesn't need to store a copy of the photo itself.

At some point we may offer an opportunity for users to opt into a Creative Commons license, and associate their photos with WOF records, but that opens up lots of questions we aren't yet prepared to address. For now Boundary Issues cares very narrowly about geotag metadata, only keeping a photograph (in `localStorage`) long enough for you to remember which place it depicts.


Boundary Issues
HI, DPHIFFER
Search for a place
GET PHOTOS
GEOTAG
Accepted formats: .jpg, .jpeg
25.061144, 121.540031
25.032225, 121.568611
Boundary Issues is sponsored by Mapzen

윤슬* or Yoonseul (https://whosonfirst.mapzen.com/spelunker/id/1158786431/)** *is a* ***public art/public space installation (http://societyofarchitecture.com/soa/project/yoonsulmanridong-reflects-seoul/) *in Seoul*

EXIF tags contain multitudes

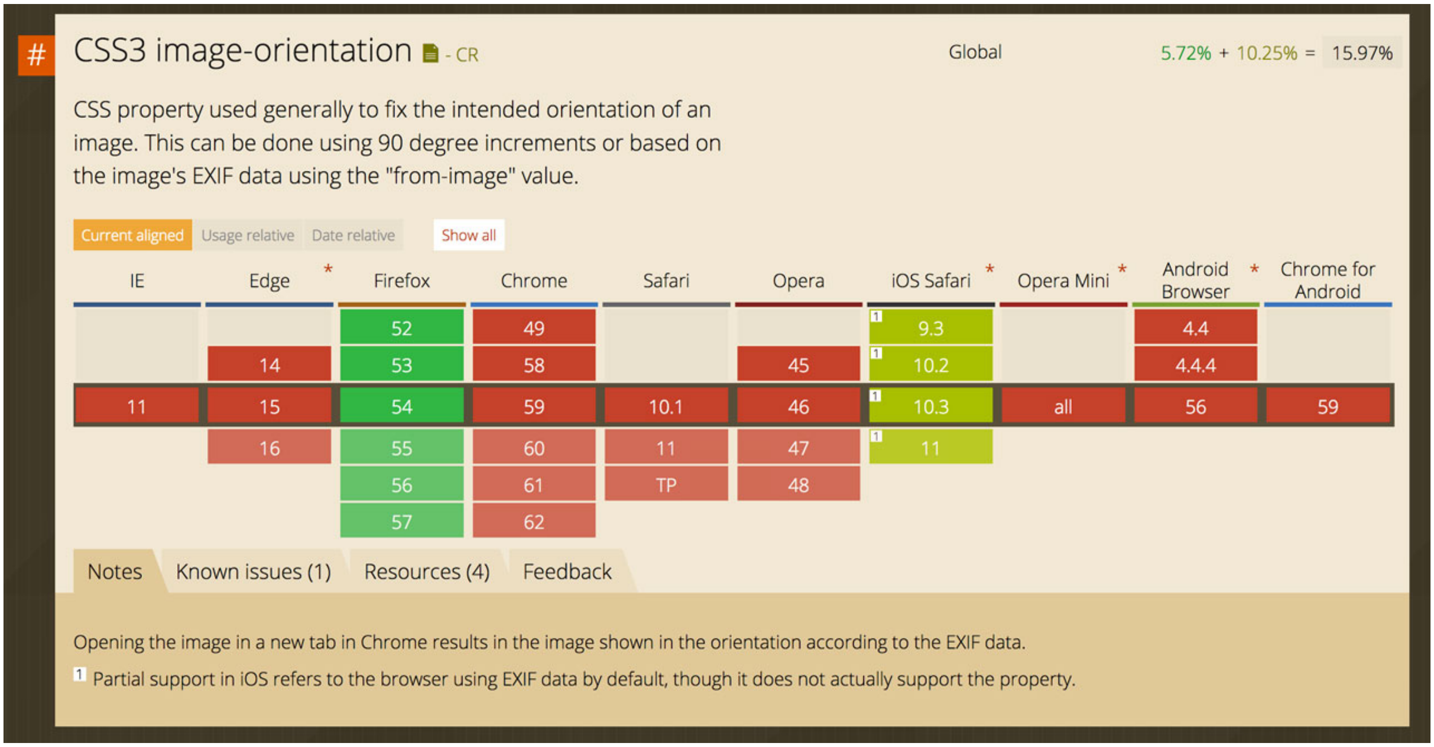
Working with photos straight off the camera has some quirks. Upon reading an image from an `<input type="file">` you get a binary `Blob` (yes, that is the **technical name (https://developer.mozilla.org/en/docs/Web/API/Blob)**) that **contains (https://en.wikipedia.org/wiki/JPEG#Syntax_and_structure)**:

- Technical metadata like width, height, tables needed to decode the image.
- A sequence of compressed bits—a “scan”—that represent the image. (In the case of “progressive scan” JPEGs there are multiple scans of increasing quality.)
- “Application-specific” metadata, including Exchangeable Image File Format (EXIF), including lots of detailed context (where/when/how it was shot) about the photo.

There are a great many things that EXIF tags can describe about a photo. The tags can tell you which model of camera produced the photo as well as details about the exposure settings like shutter speed and f-stop. In the case of (configured properly) smartphones, the EXIF tags also contain latitude/longitude/altitude metadata. I normally disable geotags on my phone camera **for privacy reasons (https://www.eff.org/deeplinks/2012/04/picture-worth-thousand-words-including-your-location)**, but they’ve been very useful for gathering venue data.

Another important tag I inadvertently encountered is the **EXIF Orientation mode (https://www.daveperrett.com/articles/2012/07/28/exif-orientation-handling-is-a-ghetto/)**. If you’ve ever seen a photo that someone posted online that looked upside-down, it’s because the image scan was stored in the upside-down order that the camera captured it, and the EXIF tag instructing the computer how to display the image right-side-up was ignored. Web browsers are still not so great applying that EXIF orientation mode (see also: my **EXIF orientation test page (https://phiffer.org/etc/exif-orientation-test/)**).

As it happens, there is a **CSS3 `image-orientation` property (https://developer.mozilla.org/en-US/docs/Web/CSS/image-orientation)** for exactly this purpose. It even includes a magical sounding `from-image` setting, which sounds like it should fix things right up. Unfortunately **nobody but Firefox (http://caniuse.com/#search=image-orientation)** supports this approach. It’s up to us web developers to rotate EXIF-oriented photos ourselves.



CSS3 image-orientation - CR

5.72% + 10.25% = 15.97%

CSS property used generally to fix the intended orientation of an image. This can be done using 90 degree increments or based on the image's EXIF data using the "from-image" value.

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		52	49			[1] 9.3		4.4	
	14	53	58		45	[1] 10.2		4.4.4	
11	15	54	59	10.1	46	[1] 10.3	all	56	59
	16	55	60	11	47	[1] 11			
		56	61	TP	48				
		57	62						

Notes Known issues (1) Resources (4) Feedback

Opening the image in a new tab in Chrome results in the image shown in the orientation according to the EXIF data.

[1] Partial support in iOS refers to the browser using EXIF data by default, though it does not actually support the property.

Compatibility is (usually) better than correctness

But wait, it gets worse! One platform, iOS, behaves differently than every other operating system (at least, so far as I've gathered). iOS arguably does the "right thing" with these tags, but it is the *only case* I know of where the EXIF tags are taken into account. Everyone else seemingly ignores the EXIF orientation entirely.

Let's say you upload a photo you took in EXIF Orientation 3 (volume buttons up) from Mobile Safari on iOS. The resulting photo will look totally normal on your iPhone, because iOS decoded the EXIF Orientation and flipped the image around for you. Everyone else on *every other platform* (including Safari on macOS) will see an upside-down version. The best part is how it *looks totally normal* to the original poster, understandably baffled by the inevitable jokes in the comments about being in the opposite hemisphere.

One additional caveat I discovered is that iOS *only* compensates for the EXIF Orientation with `<img>` tags, ignoring the EXIF orientation of background images, like non-iOS browsers. My geotagging interface switches between `<img>` and `background-image` CSS, so this detail was just one more mystery to untangle along the way.

Taipei 101 (https://whosonfirst.mapzen.com/spelunker/id/1141959909/) *as viewed from the* ***象山 Xiangshan Trail (https://whosonfirst.mapzen.com/spelunker/id/1141959907/)*** *(aka Elephant Mountain).*

Yep, CSS can fix it

The upshot of all this EXIF Orientation mode stuff is that working with photos that are upside-down or sideways is *really annoying*, making it hard to decipher whatever it was you were trying to photograph. It's important for me to get this detail right, instead of holding my device sideways/upside-down to *see the dang photo* like a chump.

After **detecting the EXIF Orientation mode (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/blob/32dbe0cd63beba4ff3f7d3d40a092fb9c8af53fd/www/javascript/mapzen.whosonfirst.geotagged.js#L228-L247)** in JavaScript and **assigning a set of CSS classes (https://github.com/whosonfirst/whosonfirst-www-**

boundaryissues/blob/32dbe0cd63beba4ff3f7d3d40a092fb9c8af53fd/www/javascript/mapzen.whosonfirst.geotagged.js#L26-L37), I used a **`transform` property (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/blob/8e6f6444f0a88859e30aa7c5fcfb181cfedb1bd9/www/css/mapzen.boundaryissues.css#L894-L920)** to rotate each image (or flip it) back into its intended position. Success! Except … in iOS, which earns a **special browser sniff (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/commit/8e6f6444f0a88859e30aa7c5fcfb181cfedb1bd9#diff-14ab22632dd21f09bf0afcbd351b8d23R788)** for not conforming. Without that special case, my CSS transform would flip an image, only to have iOS helpfully *flip the photo back upside-down again* (and then I was like (╯°□°) ╯︵ ┻━┻).

If I was processing photos on the server-side, I would prefer rotating photos by rearranging the order of bits in the JPEG scan and stripping out the EXIF orientation tag altogether. Either **`jhead –autorot flipped.jpg` (http://www.sentex.net/%7Emwandel/jhead/usage.html)** or **`exiftran –ia rotated.jpg` (https://www.kraxel.org/blog/linux/fbida/)** would neatly avoid the problem of client-side photo orientation.

Mobile telephones are really, like, *a thing* now

You may have noticed lots of people enjoy taking pictures with their mobile phones. One might say they're more like **network-connected *cameras* (https://craigmod.com/journal/photography_hello/)** than *phones* at this point. This kind of had me wondering—what if I load up a Boundary Issues geotagging page on my camera/phone, and maybe press the `<input type="file">` to launch the camera app—capturing a geotagged photo right from the (tiny) web page?

This is kind of getting back to where I started, except with a slightly different interaction than before. Instead of telling a map to gather my current location with the **`geolocation` API (https://github.com/domoritz/leaflet-locatecontrol)**, a photo could act both as a saved-for-later digital artifact and as a means to convey my current location into a WOF record.

In iOS you can totally launch the camera from a web page, and the JavaScript gets the same data Blob, but the EXIF metadata is mysteriously lacking geotags. Even if you've enabled location services for Mobile Safari, Chrome, and the Camera. Android *does* include geotags with photos taken with its version of the Chrome browser, but I've found that it occasionally omits them.

I’ve filed a **bug report with Apple (https://mapzen-assets.s3.amazonaws.com/images/geotagging-wof-venues/geotags-bug-report.jpg)** and created a **minimal test case (https://phiffer.org/etc/file-input-geotags/)**, but so far it’s hard to say *why* iOS omits geotags, or whether it’s likely to be fixed in a future release. I’ll update this post as I learn more.

Metadata is complicated

JPEG geotags, and the mixed salad of web technologies required to work with them, are tricky to get right. But the resulting user experience of capturing latitude/longitude coordinates via the camera is really seamless, once you get the settings right. This ease-of-use is burdened by the fact that geotagging metadata can be **potentially (http://www.nytimes.com/2010/08/12/technology/personaltech/12basics.html) dangerous (http://www.bbc.com/news/technology-17311702)** if you inadvertently share your location on the open web, especially for at-risk populations. (There are helpful guides online for **iOS (https://forums.imore.com/guides-how-articles/271636-guide-how-enable-disable-geotagging.html)** and **Android (https://www.youtube.com/watch?v=WeraN99NJ0U)**.)

The inertia that comes with phone camera configurations makes it easy to accidentally leave geotags enabled outside of the narrow data-gathering context. The small choices that developers and designers make about these interfaces (what are the defaults? when do you get visual cues that geotagging is enabled?) are shifting norms around what we’re comfortable sharing, with whom, and in what contexts. Software that **nudges us toward oversharing (https://www.theverge.com/2017/6/23/15864552/snapchat-snap-map-privacy-threat)** will eventually exclude and endanger those of us who have reason to worry about our location privacy (I would argue that includes *everyone*, even if they don’t realize it yet).

I want Boundary Issues to use geotagged photos responsibly, ideally reminding each user to reconfigure location services once they are done gathering venue locations. Until iOS and Android provide some explicit mechanism for this, it might make sense to send an email or Slack message reminder after each geotagging session.

I’ve been struck by how hard it is to detect which photos carry geotagged metadata and how easy it is to accidentally share my precise location online. Both iOS and Android helpfully group photo albums by location, but lack safety features like **geofencing (https://en.wikipedia.org/wiki/Geo-fence)** or warning me when I’m about to share my location metadata. There are many opportunities for improving web-based applications as well. Us web developers need to call attention to users which of their photos carry geotags, especially *before* they upload them.

Onward

Despite some challenges working with EXIF metadata, it's been fun to have a simpler mechanism for collecting new WOF venues. I now find myself photographing lots of signage, particularly those signs that include place names in both English and Korean/Chinese. I now explore the built environment thinking of which places I might want to capture for Who's On First.

I've also been doing some experiments with using the **Cordova mobile app framework (https://cordova.apache.org/)** to run a basic **Tangram map (https://mapzen.com/documentation/tangram/)**. This geotagging interface has me thinking about how an offline-friendly geotagging app might incorporate the work **Aaron (http://aaronland.info/)** has been doing on **place bookmarking (https://github.com/whosonfirst/electron-whosonfirst-bookmarks/)**. In due time all of this will be ready enough to invite more users to try it out. Please **get in touch (mailto:boundaryissues@mapzen.com)** if you're interested!

· 01 August 2017 ·

Dan Phiffer

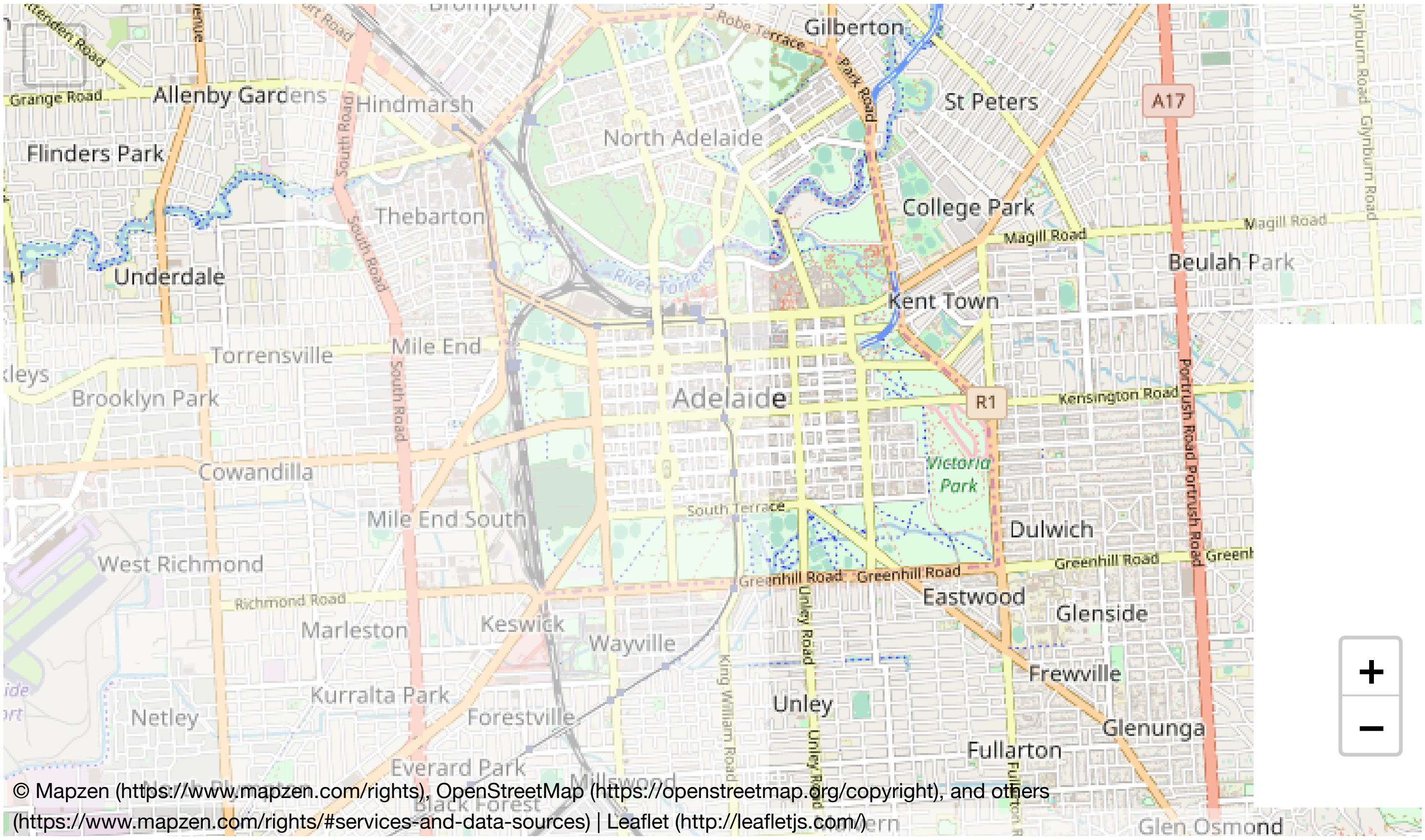
Dan is a data carpenter, which tbh is kind of a made up title. But he likes it all the same.

© 2017 Mapzen

Fun with Tangram filtering

make-your-own (/tag/make-your-own) tangram (/tag/tangram)

The great advantage of vector tiles is the ability to filter data and change the style of your map on the fly. We've all seen maps that show just metro routes and bike lanes – with Tangram, you can make these by changing just a few lines!



Gilberton
St Peters
Allenby Gardens
Hindmarsh
North Adelaide
Flinders Park
Thebarton
College Park
Beulah Park
Underdale
Kent Town
Torrensville
Mile End
Brooklyn Park
Adelaide
Cowandilla
Victoria Park
Mile End South
Dulwich
West Richmond
Eastwood
Glenside
Marleston
Keswick
Wayville
Frewville
Unley
Kurralta Park
Netley
Forestville
Glenunga
Fullarton
Everard Park
Glen Osmond
A17
R1
+
−

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

Adelaide

Compare and contrast bike networks at zoom 13, and ***open full screen (/resources/blog/filtering/bikes.html)****!*

How do you make a map like this? One way is a lot of arguing with QGIS (but we love QGIS!) Another way is with our **bike map (https://mapzen.com/blog/bike-map-v2/)**. It's simply a matter of choosing the `no-label` variant of Walkabout, filtering out roads that are not bike-related, and disabling other **layers (https://mapzen.com/documentation/vector-tiles/layers/)** like `earth` , `buildings` , `landuse` , `water` .

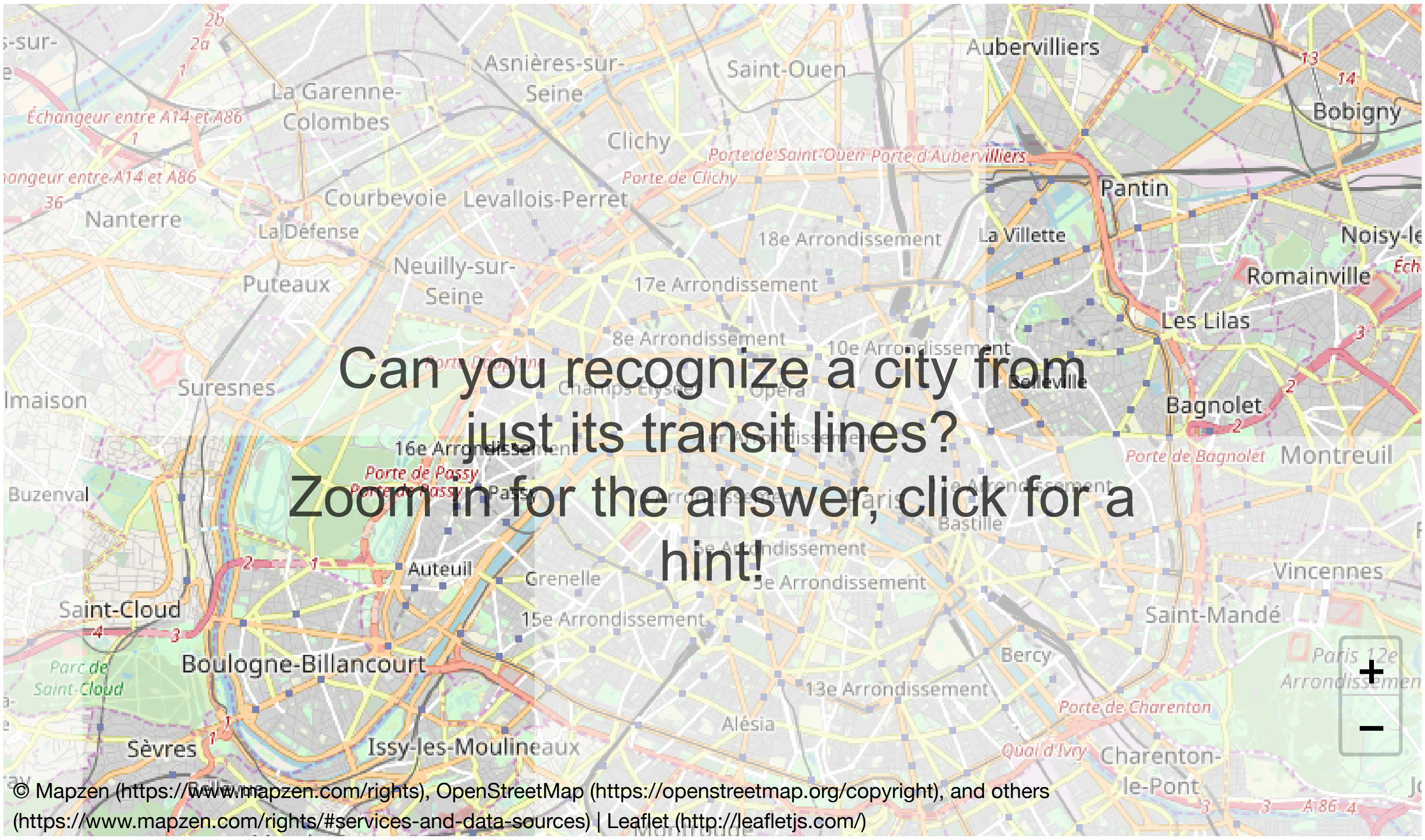
You can check out the full scene in **Tangram Play (https://mapzen.com/tangram/play/?scene=https://mapzen.com/api/scenes/22/726/resources/blank.yaml)**, but here's a quick overview.

```
import:
    https://mapzen.com/carto/walkabout-style-no-labels/5/walkabout-style-no-labels.yaml

global:
    sdk_bike_overlay: true
    sdk_path_overlay: false
    sdk_mapzen_api_key: #ADD_YOUR_MAPZEN_API_KEY_HERE http://mapzen.com/developers

layers:
    roads:
        filter:
            is_bicycle_related: true
            not:
                kind: ferry
    earth:
        enabled: false
    water:
        enabled: false
    landuse:
        enabled: false
    buildings:
        enabled: false
    places:
        enabled: false
    pois:
        enabled: false
    transit:
        enabled: false
```

For more advanced filters and data-driven mapping, Tangram can dynamically style map parameters and filter data using Javascript functions *within* the scene file. (A great example of this is our **bike map (https://mapzen.com/blog/bike-map-v2/)**, which generates colors based on infrastructure, route information and road type.)

The same holds true for transit. In fact, we turned filtering into a quiz! Can you recognize a city just by its transit network and route colors? It's harder than you might think. (Way harder.)

- click for a hint
- zoom in to see the city


Can you recognize a city from just its transit lines? Zoom in for the answer, click for a hint!

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

Aubervilliers
Saint-Ouen
Asnières-sur-Seine
La Garenne-Colombes
Clichy
Bobigny
Pantin
Courbevoie
Levallois-Perret
Nanterre
La Défense
18e Arrondissement
La Villette
Noisy-le
Romainville
Neuilly-sur-Seine
Puteaux
17e Arrondissement
Les Lilas
8e Arrondissement
10e Arrondissement
Suresnes
Bagnolet
16e Arrondissement
Porte de Passy
Porte de Bagnolet
Montreuil
Buzenval
Bastille
Auteuil
Grenelle
Vincennes
Saint-Cloud
15e Arrondissement
Saint-Mandé
Boulogne-Billancourt
Bercy
Parc de Saint-Cloud
13e Arrondissement
Porte de Charenton
Sèvres
Alésia
Issy-les-Moulineaux
Charenton-le-Pont
+
−

‹ a ›

***Open this quiz full screen to get fractional zoom!* (/resources/blog/filtering/transit_lines.html)**

For this map, while we're enabling the transit overlay and filtering out everything else in Refill, there's a twist: we are using **zoom filters (https://mapzen.com/documentation/tangram/Filters-Overview/)** to *re-enable* `places` at z12 to show city names as you zoom in, along with `roads` and `water` at z13 to give you some context once the transit labels appear.

```
import:
    https://mapzen.com/carto/refill-style/8/refill-style.yaml
global:
    sdk_transit_overlay: true
    ux_language: en # change to your language of preference, or remove to see the native
    sdk_path_overlay: false

layers:
    places:
        filter: { $zoom: { min: 12 } }
    water:
        filter: { $zoom: { min: 13 } }
    roads:
        filter: { $zoom: { min: 13 } }
    earth:
        enabled: false
    landuse:
        enabled: false
    buildings:
        enabled: false
    pois:
        enabled: false
    earth-labels:
        enabled: false
    boundaries:
        enabled: false
```

Sometimes these isolated layers verge on art. In this **Walkabout (https://mapzen.com/blog/walkabout/)** map, we've disabled everything but parks and forests.

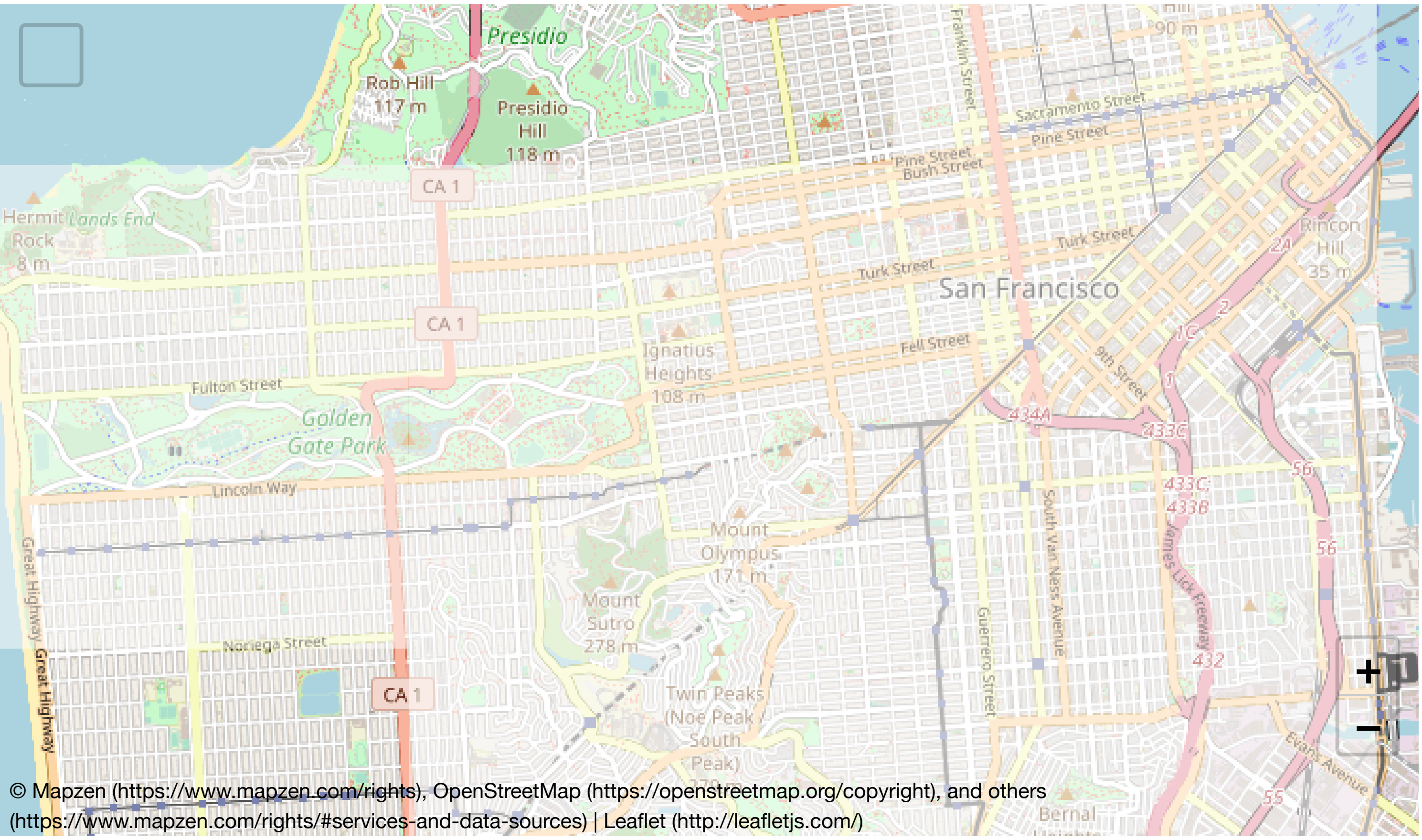
This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

```
layers:
    landuse:
        filter:
            not:
                - kind: parking
                - kind: industrial
                - kind: retail
                - kind: residential
                - kind: commercial
```

(Note that you can also collect the filter terms into an array.)

```
layers:
    landuse:
        filter:
            not:
                kind: [parking, industrial, retail, residential, commercial]
```

This reminded me of **a certain map involving islands and San Francisco (https://burritojustice.com/200ft/)**. Using some judicious regex in JavaScript functions on neighbourhood names led to a watery planet where green space and parks turn into islands on aptly-named bays and seas.


Presidio
Rob Hill
117 m
Presidio
Hill
118 m
CA 1
Hermit Lands End
Rock
8 m
Fulton Street
Golden
Gate Park
Lincoln Way
Noriega Street
Great Highway
Ignatius
Heights
108 m
Mount
Olympus
171 m
Mount
Sutro
278 m
Twin Peaks
(Noe Peak
South
Peak)
Franklin Street
Sacramento Street
Pine Street
Bush Street
Turk Street
Fell Street
San Francisco
9th Street
Rincon
Hill
35 m
90 m
South Van Ness Avenue
Guerrero Street
James Lick Freeway
Evans Avenue
Bernal
434A
433C
433B
432
56
55
2A
1C
+
−

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

San Francisco

***Open full screen, with fluid zoom!* (https://mapzen.com/tangram/view/?api=22/727#13/37.7494/-122.4265)**

```
global:
    sea_name: |
        function(feature){
            var sea_name = feature.name
            if (sea_name.match(/Mission/i)) {
                sea_name = sea_name.replace(/district/i, "Gulf")
                }
            ...
         }
...
places:
        neighbourhoods:
            neighborhood-z16:
                draw:
                    text-blend-order:
                        text_source: |
                            function(){
                                return global.sea_name(feature)
                                }
...
```

See how it was done in Tangram Play (https://mapzen.com/tangram/play/?scene=https%3A%2F%2Fmapzen.com%2Fapi%2Fscenes%2F22%2F727%2Fresources%2Fblank.yaml#13/37.7494/-122.4265) and filter your own map!

· 11 August 2017 ·

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

A "data-driven" approach to improving map-matching, Part I

routing (/tag/routing) mobility (/tag/mobility)

Since we announced our collaboration with the World Bank and more partners to create the Open Traffic platform (/blog/announcing-open-traffic), we've been busy. We've shared two (/blog/open-traffic-osmlr-technical-preview) technical (/blog/osmlr-2nd-technical-preview/) previews of the OSMLR linear referencing system. Now we're ready to share more about how we're using Mapzen Map Matching (/blog/map-matching/) to "snap" GPS-derived locations to OSMLR segments, and how we're using a data-driven approach to evaluate and improve the algorithms.

Mapzen has been testing and matching GPS measurements from some of Open Traffic's partners since development began, but one burning question remained: were our matches any good? Map-matching real-time GPS traces is one thing, but without on-the-ground knowledge about where the traces actually came from, it was impossible to to determine how close to — or far from — the truth our predictions were.

Our in-house solution was to use the **Mapzen Turn-by-Turn (/products/turn-by-turn)** routing API to simulate fake GPS data, send the synthetic data through the **Mapzen Map Matching (/products/map-matching)** service, and compare the results to the original routes used to simulate the fake traces. We have documented this process below.

You can also **view this document in a Python notebook (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/936515f7c499464e28fba28462b5d1cb2450f85b/notebooks/map_matching_part_I.ipynb)**! The Python notebook comes with Docker containers that run local instances of Valhalla, the open-source code that powers both Mapzen Turn-by-Turn and Mapzen Map Matching.

Route Generation

The first step in route generation is picking a test region, which for us was San Francisco. Routes are defined as a set of start and stop coordinates, which we obtain by randomly sampling venues from Mapzen's **Who's on First (https://whosonfirst.mapzen.com/)** gazetteer for the specified city. Additionally, we want to limit our route distances to be between ½ km and 1 km because this is the localized scale at which map matching actually takes place.

For each route, we then pass the start and end coordinates to the Turn-By-Turn API to obtain the coordinates of the road segments along the route.

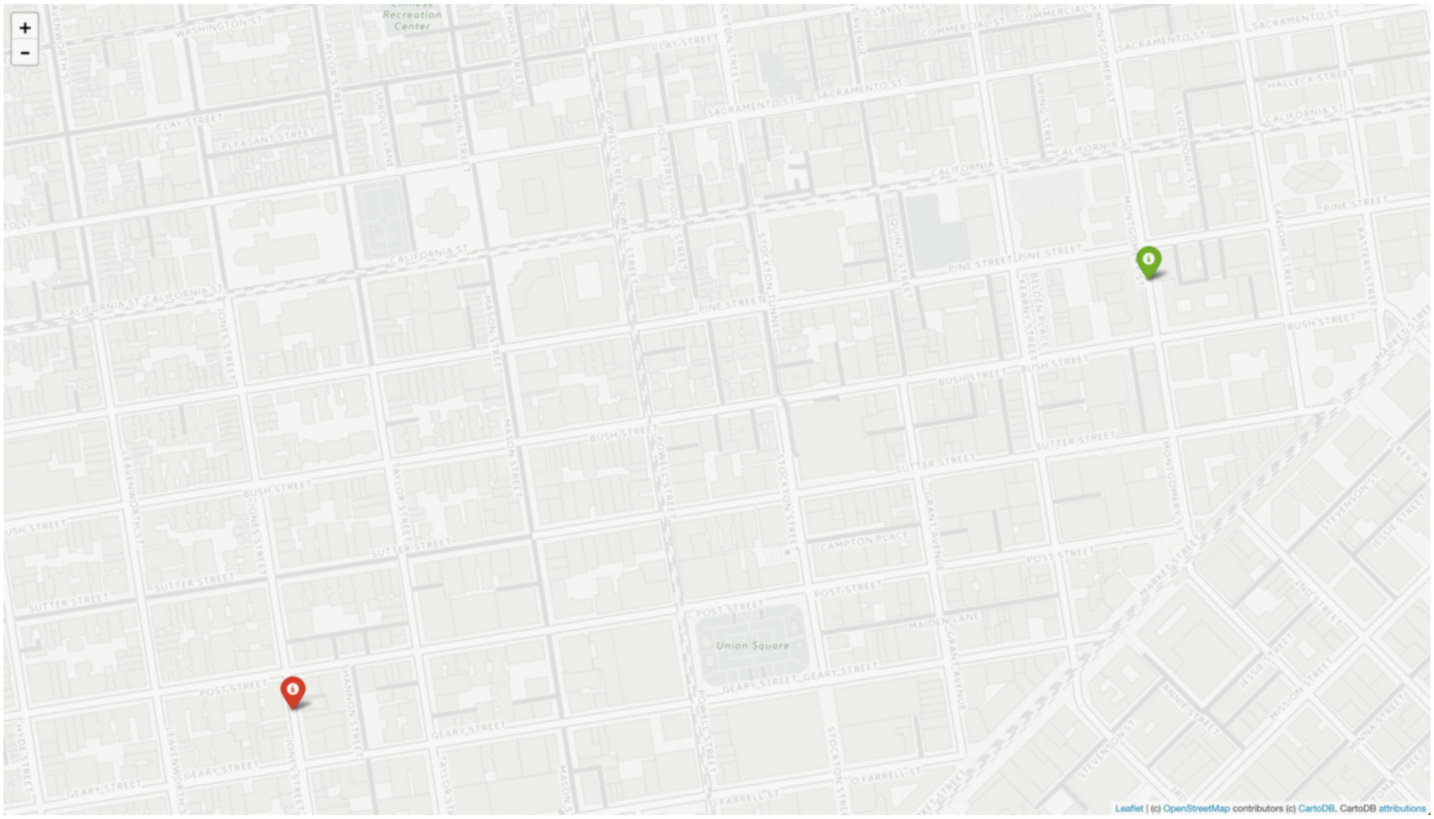
Simulating GPS data

Now that we have a set of "ground-truthed" routes, meaning route coordinates and their associated road segments, we want to simulate what the GPS data recorded along these routes might look like. This involves two main steps:

1. resampling the coordinates to reflect real-world GPS sample frequencies
2. perturbing the coordinates to simulate the effect of GPS "noise"

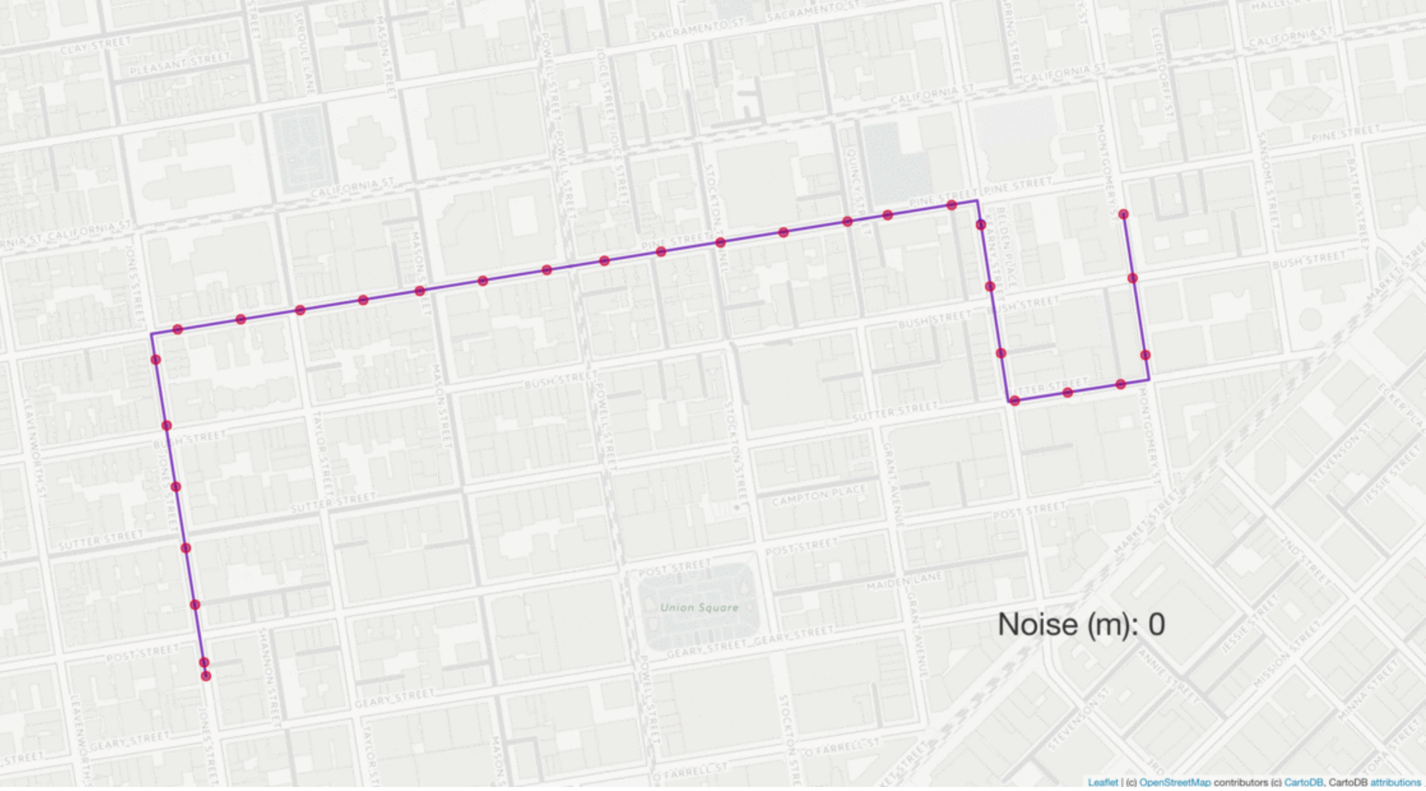
To resample the coordinates, we use the known speeds along each road segment to retain points along the route after every nn seconds.

To simulate GPS noise, we randomly sample from a normal distribution with standard deviation equal to a specified level of noise (in meters). We then apply this vector of noise to a given coordinate pair, and use a rolling average to smooth out the change in noise between subsequent "measurements", recreating the phenomenon of GPS "drift".



A route is generated from random start and end coordinates and resampled at 5 second intervals (blue dots). The resampled points are then perturbed with noise sampled from a gaussian distribution with mean 0 and standard deviation of 60. The resulting "measurements" (red dots) represent a simulated GPS trace.

Since we are interested in assessing the performance of map-matching under a variety of different conditions, we define a range of realistic sample rates and noise levels:


Noise (m): 0
Union Square
Leaflet | (c) OpenStreetMap contributors (c) CartoDB, CartoDB attributions

The same route is perturbed with varying levels of gaussian noise (red dots) with standard deviations ranging from 0 to 100 m. The resulting matched routes are shown as red lines, which only deviate from the true route at higher levels of noise.

Iterating through each our 200 routes at 5 sample rates and 21 levels of noise, we simulate 21,000 distinct GPS traces. We then pass each of these simulated traces through to the map-matching service, and compare the resulting routes against the ground-truthed, pre-perturbation route segments.

Scoring & Validation

In order to validate our matched routes, we need an effective method of scoring. Should Type I error (false positives) carry as much weight as Type II (false negatives)? Should a longer mismatched segment carry a greater penalty than one that is shorter?

We adopt the scoring mechanism proposed by **Newson and Krumm (2009) (https://www.microsoft.com/en-us/research/publication/hidden-markov-map-matching-noise-sparseness/)**, upon **whose map-matching algorithm the Valhalla Meili map-matching service is based (https://github.com/valhalla/valhalla/blob/master/docs/meili/algorithms.md)**:

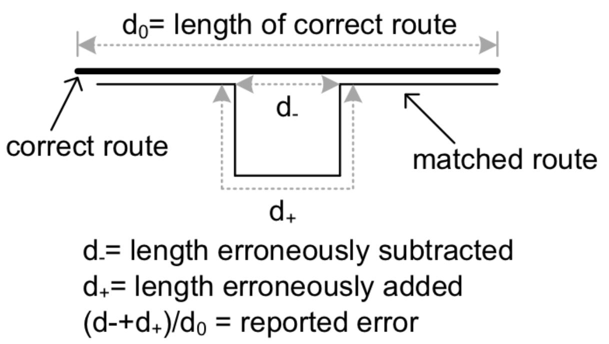
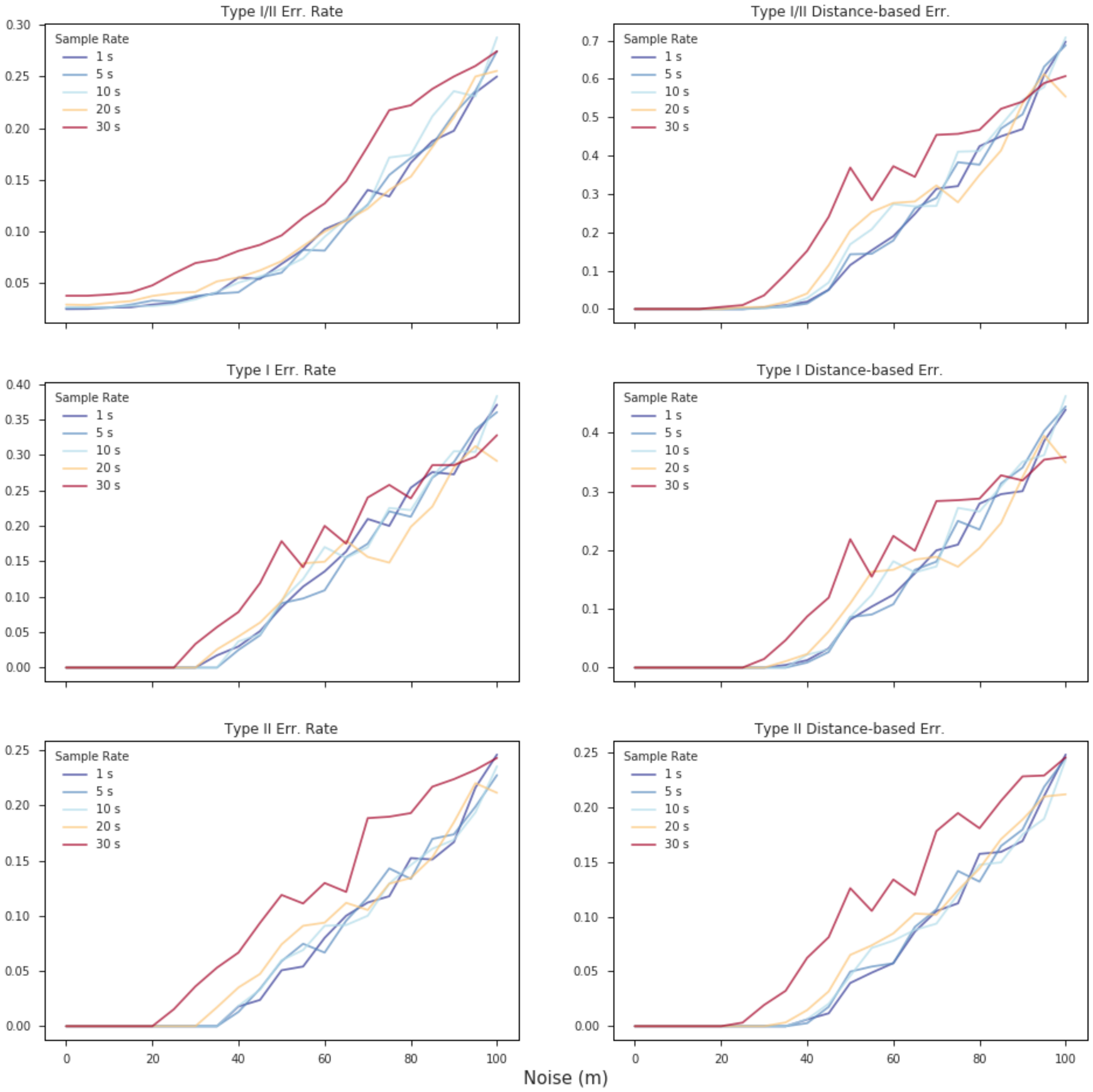

d0= length of correct route
correct route
d-
matched route
d+
d-= length erroneously subtracted
d+= length erroneously added
(d-+d+)/d0 = reported error

Figure 6: This illustrates how we measured the error between the correct route and the route from map matching.

From Newson and Krumm (2009)

In the above schematic, d_+ refers to a false positive or Type I error, while d_- represents a false negative, or Type II error. The final reported error is a combination of both types of mismatches, weighted by their respective lengths.

The graphs below represent the median scores for 6 error metrics applied to each of our 21,000 routes, broken down by sample rate and noise level. Plots in the left column are based solely on error rate, i.e the percentage of Type I, Type II, or Type I/II mismatches. The right-hand column shows the same metrics as the left, but weighted by segment length. The top right plot thus represents the metric used by Newson and Krumm, and the two plots below it represent the same value broken out by error type.



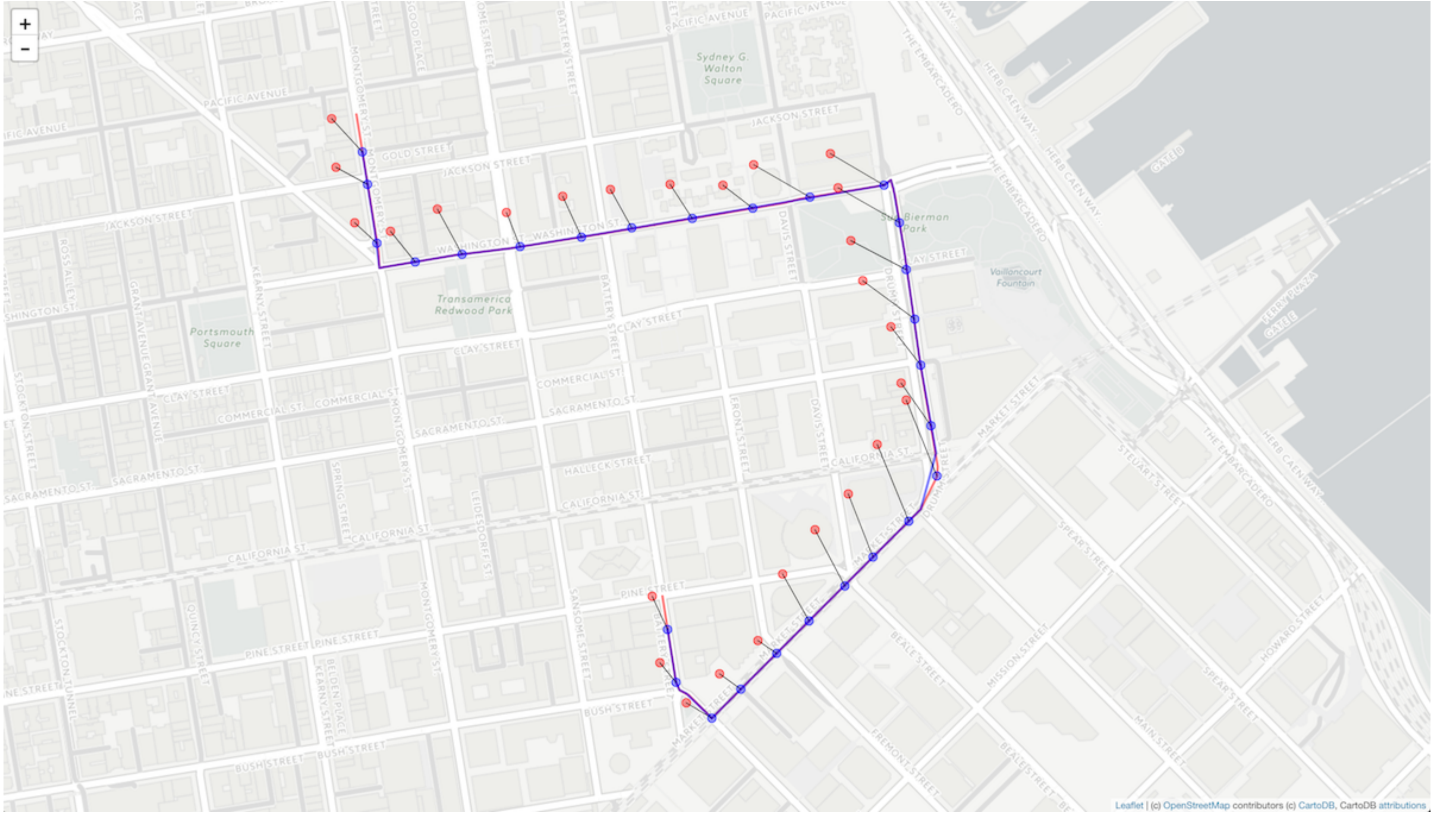
Results

There is a lot of valuable information to be gleaned from our validation process. The main takeaways are summarized below:

1. Below 50 m of noise, we obtain a median match accuracy of 90% or better.
2. Clearly the positional accuracy of GPS data (vis-a-vis noise level) has a huge impact on rate of mismatch. However, there is little to no impact on error rates at noise levels below 20 m.

3. Sample rate also has a significant effect on performance. What that effect is, however, depends significantly on the positional accuracy of the data. Higher frequency sample rates clearly perform better at lower levels of noise, but there exists a very consistent point of inflection (~80-90 m) about which the performance of low sample rate GPS traces (red) improves relative to that of the higher sample rates (blue). These results, consistent with the findings of Newson and Krumm (2009), suggest that lower sample rates tend to be more robust against mismatch error at higher levels of noise.
4. There does not appear to be a significant difference in the trends between the distance-based error metrics and those that do not consider segment length.
5. Similarly, the Type I, Type II, and Type I/II error rates very closely track one another, although the pattern of inflection described in (3) appears more pronounced in the Type II-based metrics.

Ultimately, our validation allows us to state with a greater measure of confidence that our map-matching is working as intended, at least in San Francisco:


+
-
Leaflet | (c) OpenStreetMap contributors (c) CartoDB, CartoDB attributions

A perfect match on a route with 80 m of noise (5 s sample rate)

In the next installment of this series, we will see how we can actually use these error metrics to optimize the performance of our map-matching algorithm based on the quality of the data made available to us.

· 15 August 2017 ·

Max Gardner

Max is a summer 2017 intern on the Mapzen Mobility team. A PhD student at UC Berkeley, he will continue to support the Cavs even if LeBron leaves Cleveland again because, after all, Max left too.

© 2017 Mapzen

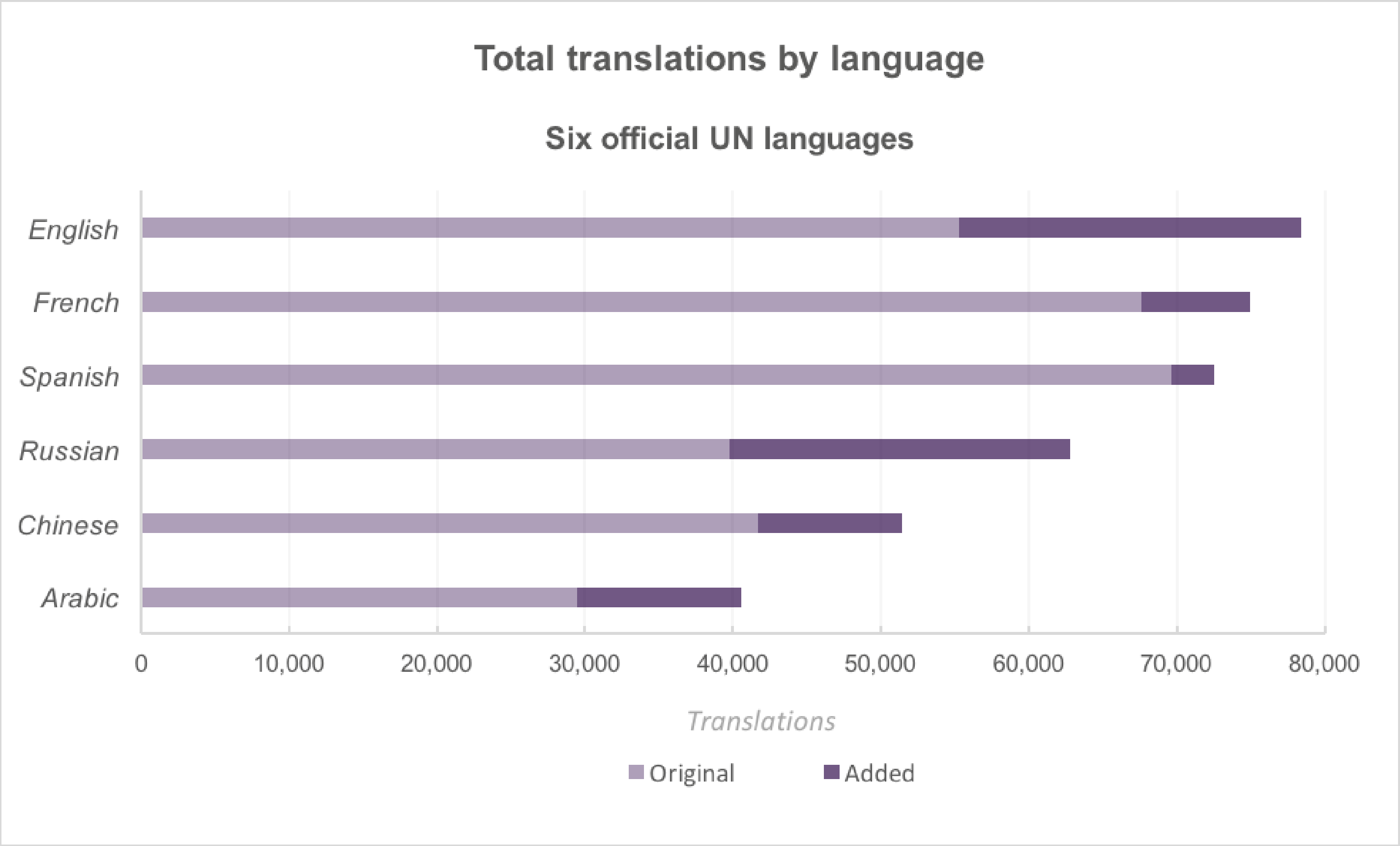
Increasing Name Translations in Who's On First

whosonfirst (/tag/whosonfirst) **data (/tag/data)** **interns (/tag/interns)**

This summer, Mapzen embarked on a large project to quantify and visualize name translation coverage in **Who's On First (https://whosonfirst.mapzen.com/)**, and to add new translations from **GeoNames (http://www.geonames.org)** by walking the concordance between the two datasets. Much of the work was done by Nathaniel Douglass, our summer intern on the Data team.

Who's On First is Mapzen's big list of places, each with a stable identifier and descriptive properties. These places represent various **placetypes (https://github.com/whosonfirst/whosonfirst-placetypes)** like **`country` (https://whosonfirst.mapzen.com/spelunker/placetypes/country/)**, **`region` (https://whosonfirst.mapzen.com/spelunker/placetypes/region/)**, **`locality` (https://whosonfirst.mapzen.com/spelunker/placetypes/locality/)**, and even **`marinearea` (https://whosonfirst.mapzen.com/spelunker/placetypes/marinearea/)**.

In addition to having every place documented, an overarching goal for Who's On First is to catalogue all **names (https://github.com/whosonfirst/whosonfirst-names)** in all the languages.



Above: The number of new translations added this summer, for the six major world languages used by the United Nations. For example, English gained 23,115 new name translations, while Spanish gained 2,916 new name translations.

Previous work around names…

Last summer, **Olga** wrote about adding **name translations from Wikidata (https://mapzen.com/blog/wikipedia-data/)** to Who's On First. That Wikidata effort add nearly **two million** localized names for over **135,000** Who's On First records, joining earlier names from **Geoplanet (http://developer.yahoo.com/geo/geoplanet/)**. **Nathaniel Douglass** has continued that work this summer. Now over to him!

Tackling the issues

Our goal, and specifically, my internship task, was to add *more* of these name labels for *more* places. But there are so many names!

Not only do I want to ensure that Who's On First has new and accurate `name:` properties, but I also needed to ensure the following is true for Who's On First records:

- The `name:` properties are using correct ISO language codes.
- The `name:eng_x_preferred` property is present if the record has an existing English `wof:name` .
- Concordance values for GeoNames are set, which can yield new `name:` properties.

Where is my starting point?

It’s safe to say that adding all missing names is much too large of a task for one humble intern to accomplish within two and a half months. So, I needed to ask myself one question:

> “Where do I start?”

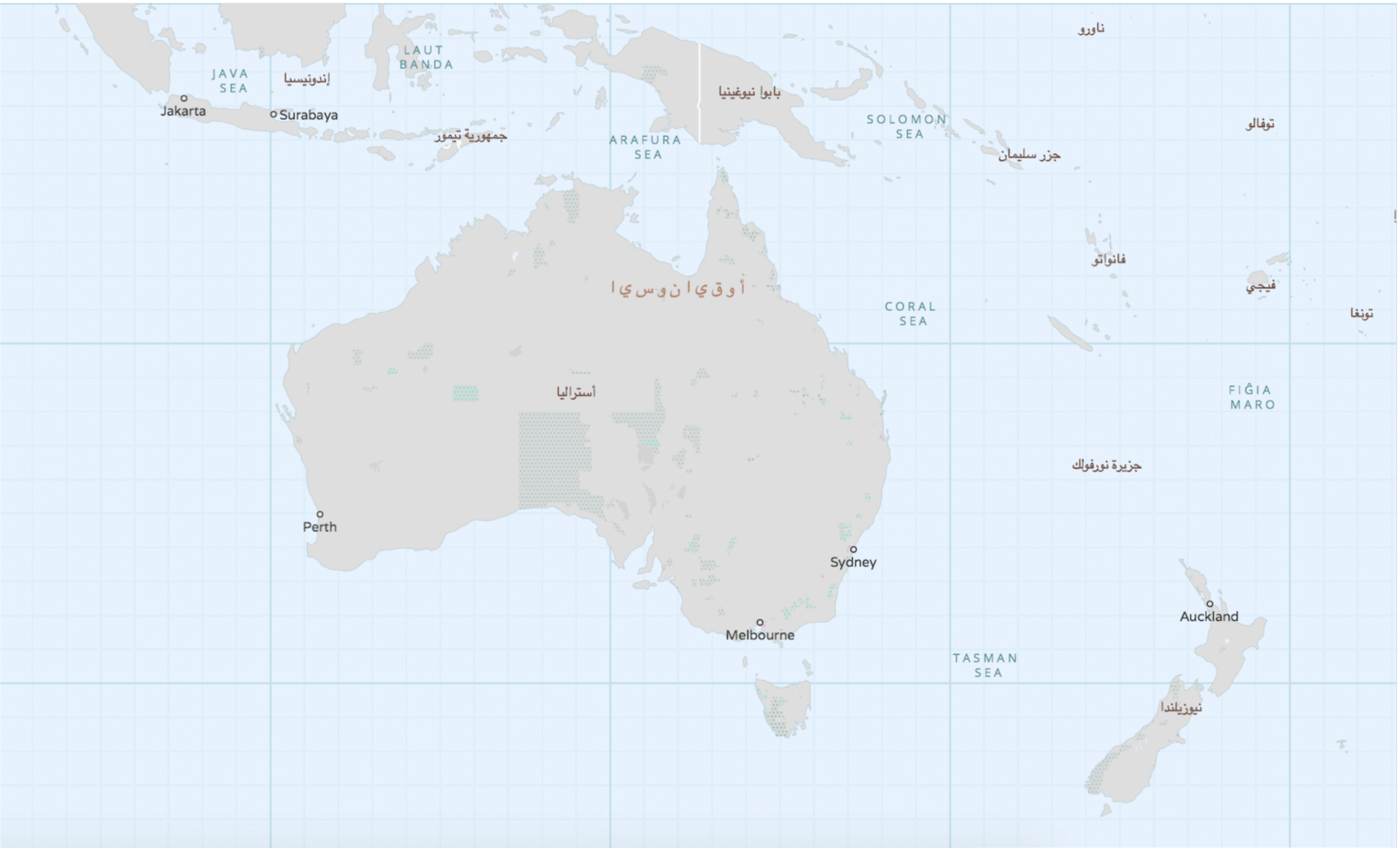
Before running name import scripts and importing new data, it was important for me to visualize the amount of names present. I did this by programmatically creating charts using Python and Excel.

The nitty gritty…

The International Organization for Standardization (**ISO (https://www.iso.org/about-us.html)**) maintains standardized language codes called the “**ISO 639 Standard (https://en.wikipedia.org/wiki/ISO_639)**”. The **ISO 639-2 language codes (https://en.wikipedia.org/wiki/ISO_639-2)** are used for the principal language identifier in Who’s On First’s `name:` properties (for more information on our use of ISO language codes and **RFC 5646 (http://www.rfc-editor.org/rfc/rfc5646.txt)**. See our **properties (https://github.com/whosonfirst/whosonfirst-properties)** and **names (https://github.com/whosonfirst/whosonfirst-names)** repositories), for example:

```
"name:eng_x_preferred:[
    "North America"
],
"name:lat_x_preferred:[
    "America Septentrionalis"
],
"name:sin_x_preferred:[
    "උතුරු ඇමෙරිකාව"
],
"name:spa_x_preferred:[
    "América del Norte"
],
```

The above `name:` properties show “**North America (https://whosonfirst.mapzen.com/spelunker/id/102191575/)**” in English, as well as the Latin, Sinhalese (Sinhala), and Spanish translations.


ناورو
LAUT BANDA
JAVA SEA
إندونيسيا
بابوا نيوغينيا
Jakarta
Surabaya
SOLOMON SEA
توفالو
جمهورية تيمور
ARAFURA SEA
جزر سليمان
فانواتو
أوقيانوسيا
فيجي
CORAL SEA
تونغا
أستراليا
FIĞIA MARO
جزيرة نورفولك
Perth
Sydney
Auckland
Melbourne
TASMAN SEA
نيوزيلندا

Above, the `continent` of **Australia (https://whosonfirst.mapzen.com/spelunker/id/85632793/)**, is shown in Mapzen’s **Bubble Wrap (http://tangrams.github.io/bubble-wrap/#4/-28.73/144.62)** basemap style. You’ll notice the Australia label is set to the default language (in this case, English), and, when we switch the name label to Arabic, we see the label change to `أستراليا` . *(You can do this in Bubble Wrap by toggling the “language” drop-down in the Controls menu, or by* ***defining (https://mapzen.com/blog/languages-of-india/)*** *the* `uxlanguage` global in Tangram)_.

Less is more (at least when it comes to names)

While ISO 639-2 catalogues over 450 language codes, I am mostly interested in the most-spoken languages of the world. Lucky for me, **Tilezen (https://github.com/tilezen/)** (which powers Mapzen Vector Tiles) lists **20 common languages (https://github.com/tilezen/vector-datasource/blob/master/SEMANTIC-VERSIONING.md#languages-are-not-versioned)** based on the six official languages of the United Nations, as well as the most commonly referenced languages on **Wikipedia (https://en.wikipedia.org/)**. A breakdown of those languages, below:

Language (*ISO-639-2*)			
Arabic (*ara*)	French (*fra/fre*)	Italian (*ita*)	Russian (*rus*)
Bengali (*ben*)	German (*deu/ger*)	Japanese (*jpn*)	Spanish (*spa*)
Chinese (*zho/chi*)	Greek (*ell*)	Korean (*kor*)	Swedish (*swe*)
Dutch (*nld/deu*)	Hindi (*hin*)	Polish (*pol*)	Turkish (*tur*)
English (*eng*)	Indonesian (*ind*)	Portuguese (*por*)	Vietnamese (*vie*)

Who’s On First provides names in more than 60 languages, but not all languages have equal coverage. There are 10 languages with great coverage (more than 50,000 features translated), and another 25 language with good coverage (at least 10,000 features translated).

I want to ensure that the name translations are present in the above languages, but I also need to verify translations are available at different zoom levels in our map house styles. Zoom levels are present in Who’s On First records as the `mz:min_zoom` and `mz:max_zoom` properties and indicate when a label appears on a map. By querying records by their `mz:min_zoom` and `mz:max` zoom property values, I will get a more comprehensive understanding of our name translation coverage.

Less is more (at least when it comes to features)

When I was looking at online map providers and print atlases at the beginning of my internship, I noticed that it’s often just the low-zooms that are translated into all the languages, with native language names (and sometimes a romanized or “English” name) being dominant at mid- and high-zooms.

Natural Earth (http://naturalearthdata.com/) provides a curated list of a few thousand (instead of million!) important places tuned for low-zoom cartography. Using a pre-release version of the upcoming Natural Earth version 4 data (stored into `mz:min_zoom` , `mz:max_zoom` properties in Who’s On First), I was able quantify the entire gazetteer *and* a smaller set of features to focus on the features used to label maps at low-zooms.

Let’s dive into all that…

Grouping records

I crawled through the `whosonfirst-data` repository to gather statistics about each record for specific placetypes and zoom levels. I then grouped records into the following categories:

- By placetype
- By placetype, then by language
- By placetype, then by zoom
- By placetype, then by zoom, then by language

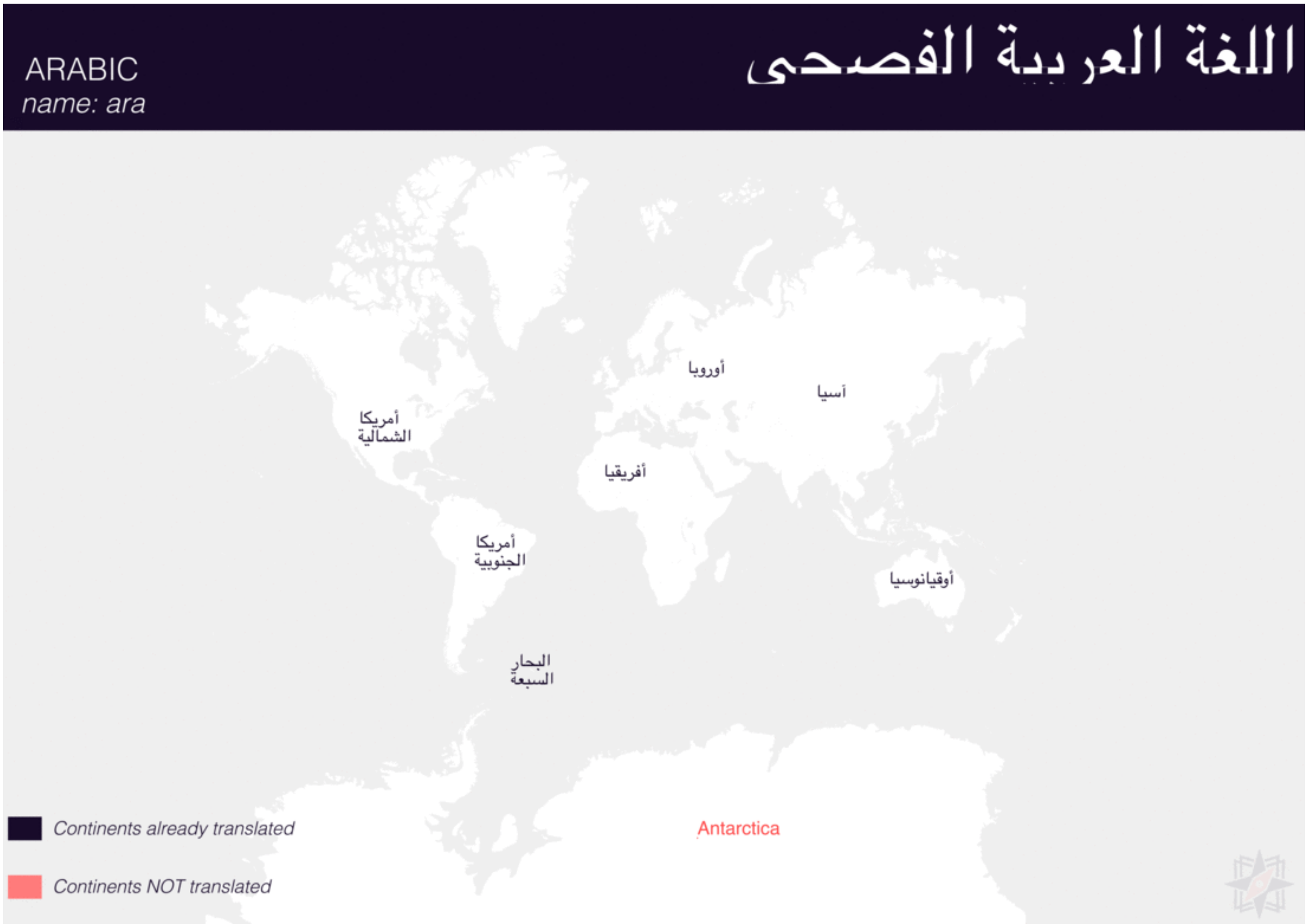
For this work, I focus only on the following placetypes: `country` , `dependency` , `disputed` , `region` , `locality` , `continent` , `ocean` , and `marinearea` - the features visible at low- and mid-zooms on a map and most useful for point-in-polygon lookups. Other placetypes like `county` , `neighbourhood` , and `venue` were left for another day.

After obtaining the `wof:name` , `wof:placetype` , `mz:min_zoom` , `mz:max_zoom` , and the existing `name:` properties, I flagged and grouped records by the categories above. The output was a series of csv files, which I crafted into visualizations.

Maps!

Using **QGIS (https://www.qgis.org/)**, I was able to import and style featuress to show which had translations (dark purple labels) and which didn't (red labels). The map below is categorized by placetype and depicts Who's On First records that have translations for the twenty languages of interest:

Continents


ARABIC
name: ara
اللغة العربية الفصحى
أوروبا
آسيا
أمريكا الشمالية
أفريقيا
أمريكا الجنوبية
أوقيانوسيا
البحار السبعة
Antarctica
■ Continents already translated
🟥 Continents NOT translated

Here are additional graphics for the **ocean (https://mapzen-assets.s3.amazonaws.com/images/names-in-wof/maps/ocean.gif)**, **disputed (https://mapzen-assets.s3.amazonaws.com/images/names-in-wof/maps/disputed.gif)**, **marinearea (https://mapzen-assets.s3.amazonaws.com/images/names-in-wof/maps/marinearea.gif)**, and **region (https://mapzen-assets.s3.amazonaws.com/images/names-in-wof/maps/region.gif)** placetypes.

Holding Hands with Open Data

While I was busy crunching numbers and charting, **Stephen** went prospecting for additional names.

Existing resources

Luckily, the open-source community has compiled many name translations. We can tap into these translations through concordances that exist in Who's On First records. Websites like **Geonames (http://www.geonames.org/)**, **Wikidata (https://www.wikidata.org/)**, **Geoplanet**

(http://developer.yahoo.com/geo/geoplanet/), and **Natural Earth (http://www.naturalearthdata.com/)**, have exactly what we are looking for.

For each of these sources, a particular record will have a unique identifier that Who's On First holds hands with. This allows us to augment our existing name translations. An important caveat — just because Who's On First has a concordance value does not always mean we can extract data from that source. Often times, Who's On First already has a name translation, or the source may not have any new names (or any at all).

Mapzen uses Who's On First data in several services including **Vector Tiles (https://mapzen.com/projects/vector-tiles/)** which powers parts of the Bubble Wrap map shown above (along with **OpenStreetMap (http://www.openstreetmap.org)** data) and in **Search (https://mapzen.com/products/search/)** for both name lookups and reverse geocoding.

GeoNames

GeoNames **maintains an alternative names file (http://download.geonames.org/export/dump/)** that contained roughly twelve million rows, each with data for a single unique and stable GeoNames identifier.

Using the identifier crosswalk in the Who's On First concordance list, and after mapping the 2-character language codes used the GeoNames file to the 3-character ISO 639-2 codes used in Who's On First, Stephen was able to pull out any names (and abbreviations) that Who's On First did not already have.

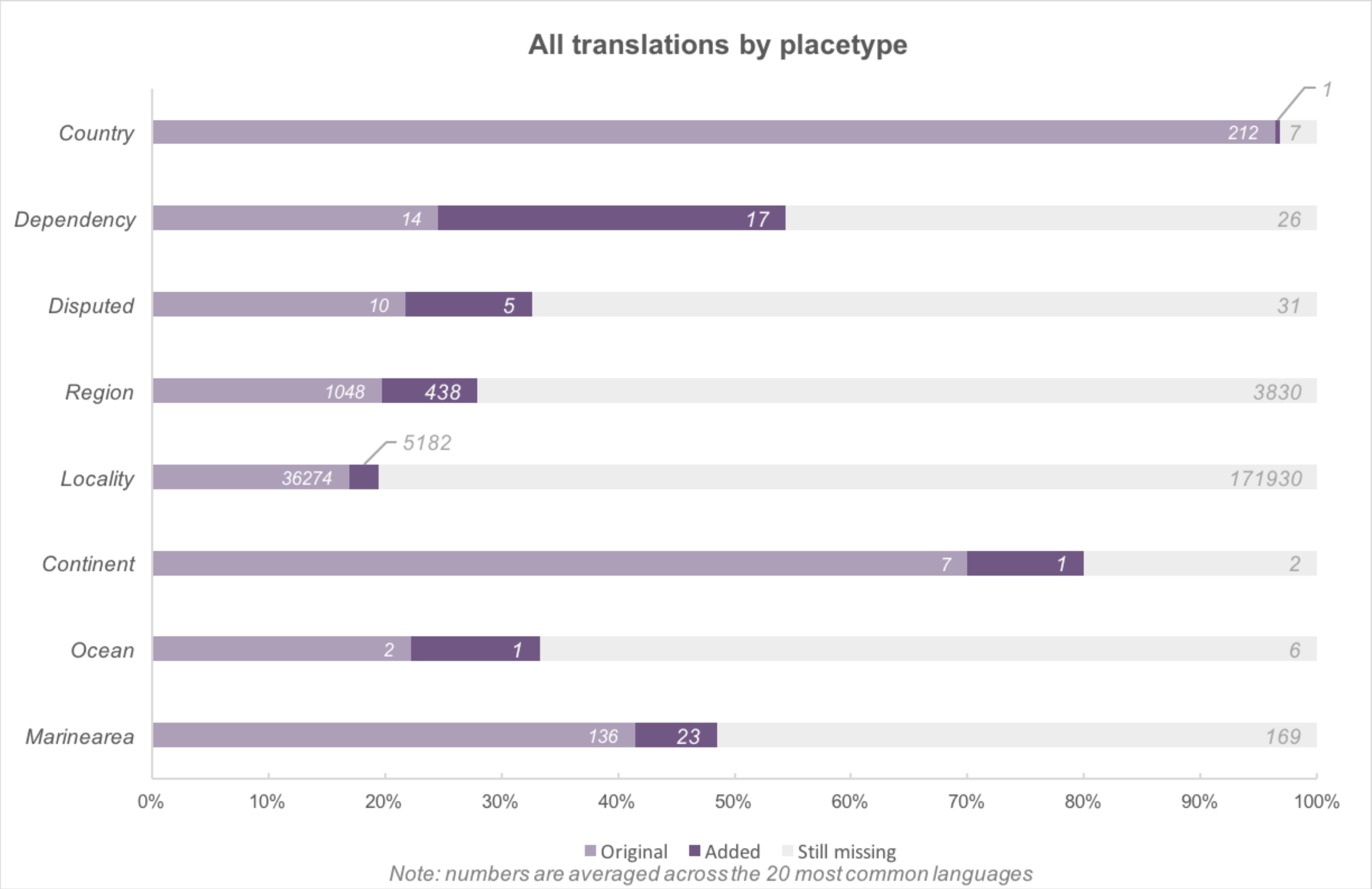
Results

After Stephen added the new `name:` data to Who's On First (and I cleaned up some names manually), it was time for me to see what was gained! To do this, I re-ran the same script used to generate charts on the repository to update charts and visualize the translation gains.

In total, over **113,300** records were updated to include over **255,000** new, unique name translations! While the majority of updated records gained one or two new name translations, this work filled large gaps in coverage for records that had zero name translations to begin with. For instance, a record for **Easter Island (https://raw.githubusercontent.com/whosonfirst-data/whosonfirst-data/master/data/890/517/533/890517533.geojson)** gained over **100** new name translations!

From Occitan to Catalan to Persian, you're now able to browse our records through our **Spelunker tool (https://whosonfirst.mapzen.com/spelunker/)** to view these new names.

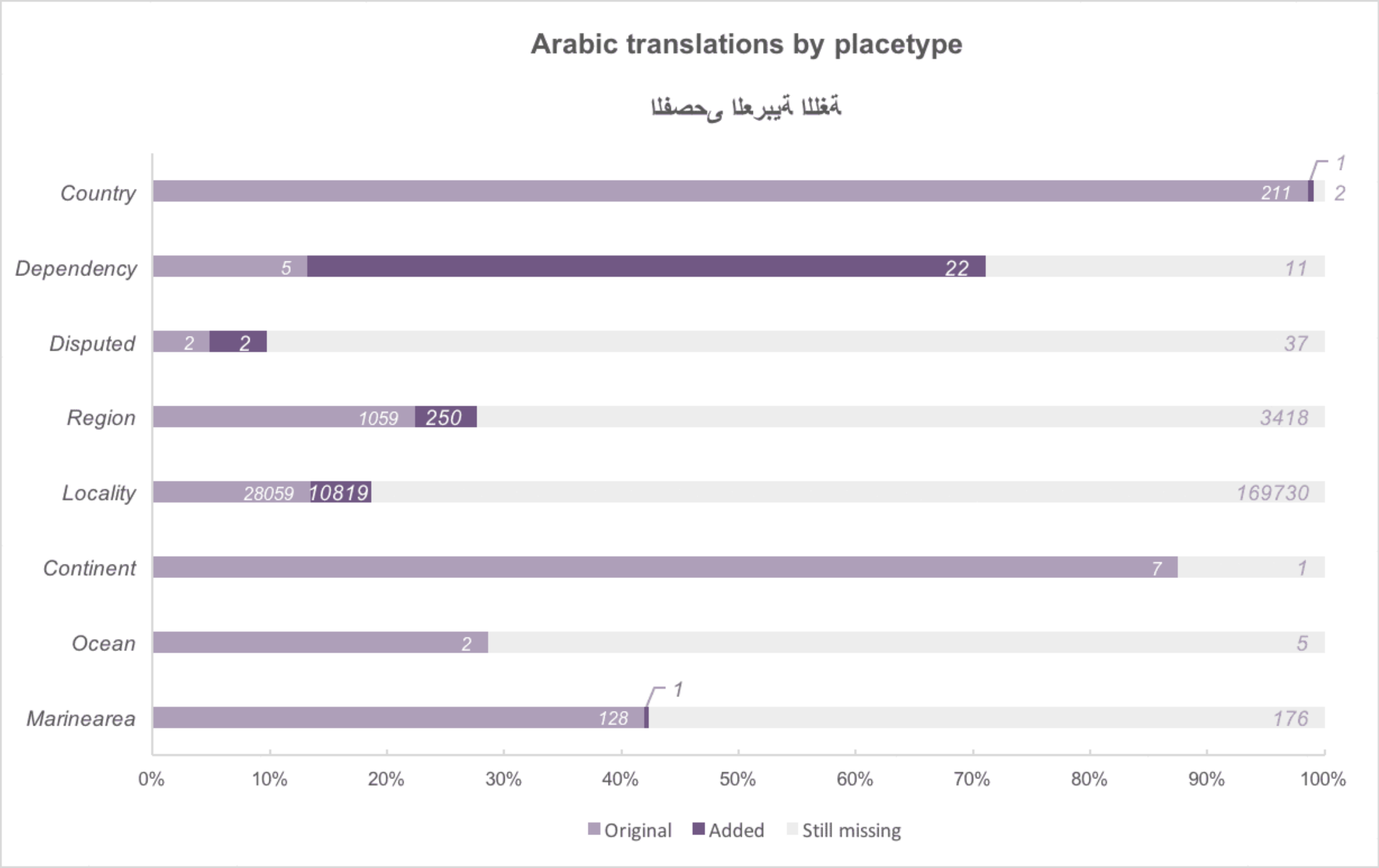
By placetype (overall)



The chart above shows the total increase in translated names by each placetype. The light purple represents what I started with in the beginning of the summer, the darker purple is how many names were added, and the grey bar represents what is still missing. The numbers represent an average for all 20 languages.

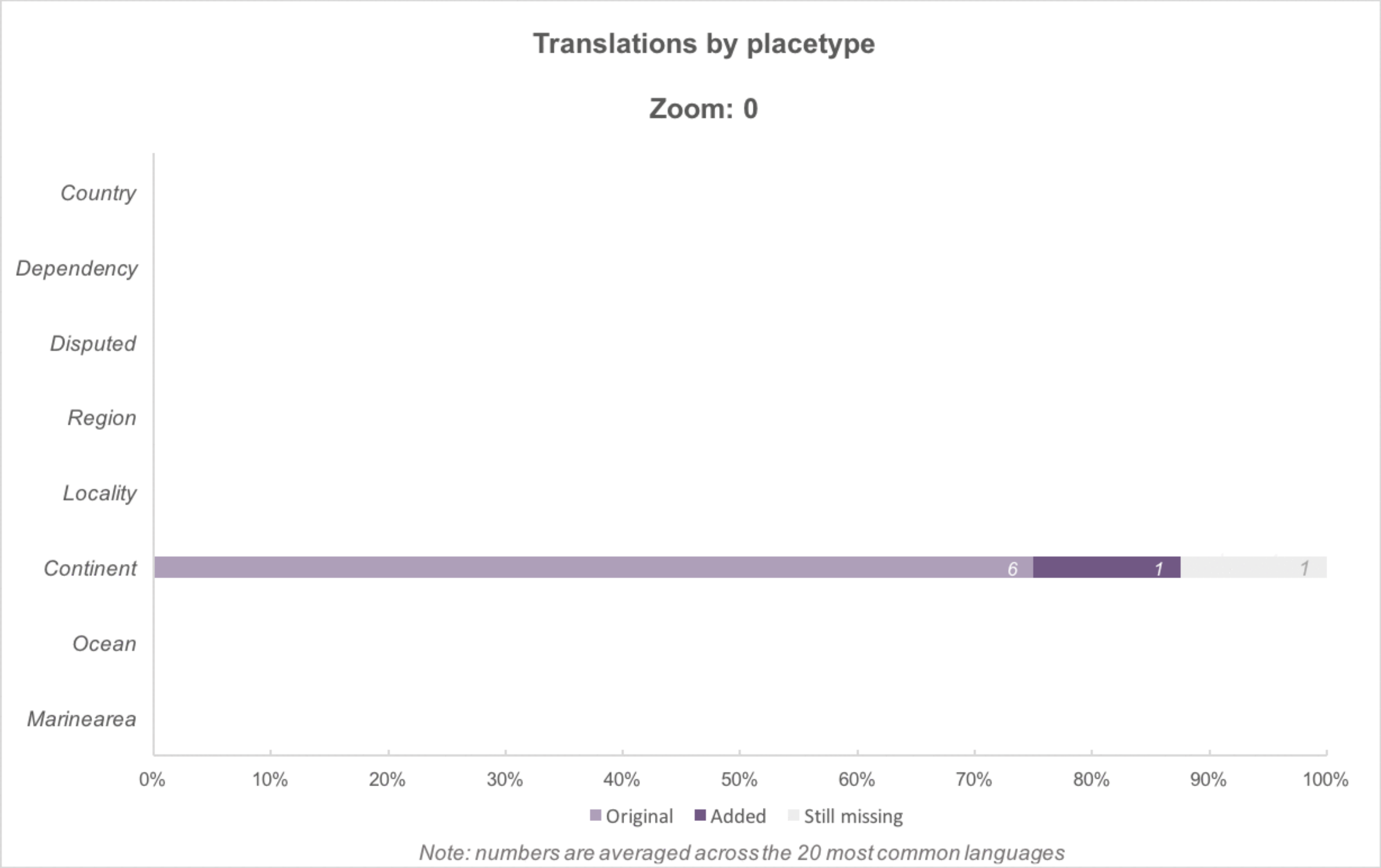
Note: You'll notice a discrepancy in the number of `country` *records when comparing pre and post charts. The overseas insular regions of France (**Reunion (https://whosonfirst.mapzen.com/spelunker/id/85671199/)**, **Guadalupe (https://whosonfirst.mapzen.com/spelunker/id/85671209/)**, **Martinique (https://whosonfirst.mapzen.com/spelunker/id/85671191)**, **French Guiana (https://whosonfirst.mapzen.com/spelunker/id/85671195/)**, and **Mayotte (https://whosonfirst.mapzen.com/spelunker/id/85671203/)**) were reclassified from* `country` *records to* `region` *records in Who's On First this summer, but don't worry - these records gained new name translations, too!*

By placetype, then by the six languages used by the United Nations (comparison)



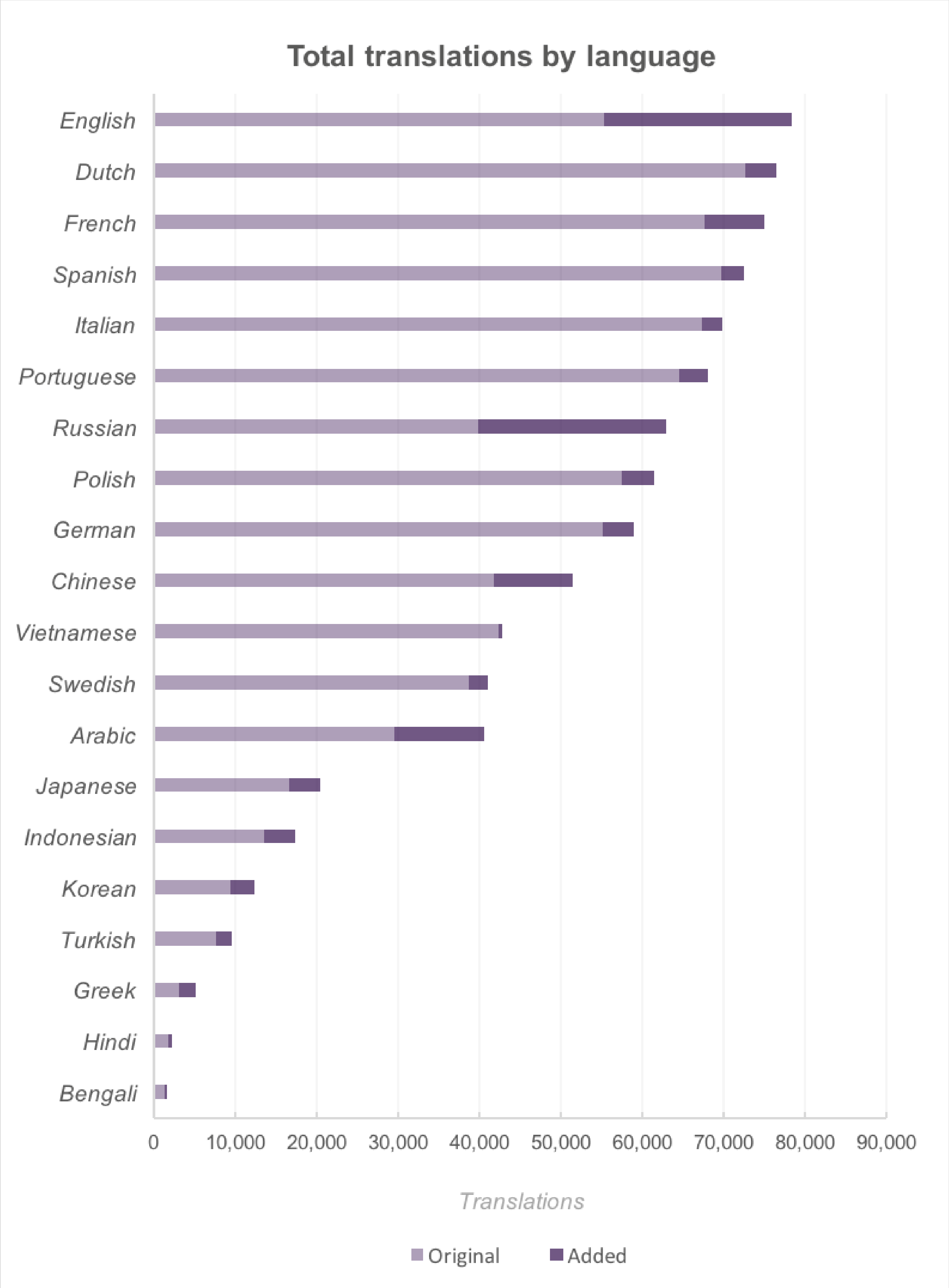
Above: The added translations, by placetype, for the six official UN languages. Work done for English translations resulted in 100% coverage across four placetype categories!

By placetype, then by zoom level (comparison)



Above: Added translations, by placetype, for zoom levels 0 to 8. There are fewer placetype categories and a higher percentage of translated names at the lower zoom levels. At the highest zoom levels, translation percentages drop as more features become present. (Other charts in this post are equivalent to zoom 16.)

By language (overall)


Total translations by language
English
Dutch
French
Spanish
Italian
Portuguese
Russian
Polish
German
Chinese
Vietnamese
Swedish
Arabic
Japanese
Indonesian
Korean
Turkish
Greek
Hindi
Bengali
0
10,000
20,000
30,000
40,000
50,000
60,000
70,000
80,000
90,000
Translations
Original
Added

Above: This chart shows the total number of features with translations for all 20 languages we focused on. Dutch appeared to be the highest translated language before the summer, with an additional 3,800 name translations added, but was beat by English which saw larger gains.

What Did I learn?

Before beginning this internship, I hadn't the slightest idea how I would translate millions of records into multiple languages. In fact, it was a rather stressful first week mauling over where to begin. However, with the help of some amazing colleagues I was able to narrow the problem down, and approach this more methodically. I learned first hand that the best way to eat a whale is one bite at a time!

This summer I gained some valuable experience writing Python scripts, dealing with large datasets, and create meaningful charts to help the data team. Aside from a single Python course last semester, I had never used programming in this way and the real world application has opened my eyes to the possibility of making this a career. Even though it wasn't fancy, I spent ample time perfecting how to generate charts manually in Excel that would visualize the work. While there are many automated charting libraries out there, this allowed me to work closely with the data and fine tune how best to visualize it using a tool I was familiar with.

Additionally, I was introduced to using GitHub in a professional setting, which has been a frustrating, yet rewarding experience. I have come to understand version management, pull requests, and how to deal with merge conflicts, without pulling out all of my hair (beard hair to be specific).

I would like to thank my manager Nathaniel Vaughn Kelso for his vision regarding this project, as well as his confidence in my abilities. I also must thank Stephen Epps for his constant support and dedication towards my success. This project wouldn't have been possible without their guidance and encouragement. For that, I am very grateful.

What's Next?

As you can see in the charts, we still have areas of improvement in Who's On First. We are now able to better visualize translations in Who's On First using these charts, as well as a new feature in our Spelunker to filter by translations (see: for example, see the translations filter in the sidebar to find descendants of Africa in our **Spelunker (https://whosonfirst.mapzen.com/spelunker/search/?q=africa)**).

Mapzen has more concordance work, name imports, and one-off fixes planned, with an emphasis of increasing coverage by language, placetype, and zoom-level.

If you have names you would like to add to a record, please feel free to file a new issue in the `whosonfirst-data` **(https://github.com/whosonfirst-data/whosonfirst-data/issues/new)** repository or **drop us a line (mailto:hello@mapzen.com)**!

Photo Credit: ***quiddle. (Flickr) (https://flic.kr/p/ewKKu)***

· 22 August 2017 ·

Nathaniel Douglass

Nathaniel was a summer 2017 intern on the Mapzen data team. He is a Geography undergrad at Humboldt State University and has an unhealthy obsession with maps and kimchi.

Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.

© 2017 Mapzen

New and improved bike routing, with low stress options

routing (/tag/routing) interns (/tag/interns) mobility (/tag/mobility)

Bicyclists ride to commute, to exercise, for sport, for leisure. But no matter the reason, most cyclists will ride on the road at some point, and when they do, they'll think about the safety and comfort of their route.

Should you take a longer route to ride on a road with a dedicated cycling lane? Do you take a left turn to avoid those difficult hills? Different cyclists will make different choices, as the ideal route for one bicyclist may not be so ideal for another.

Valhalla, the open-source routing engine that powers **Mapzen Turn-by-Turn (/products/turn-by-turn)**, has always had customization parameters. While these include `use_roads` and `use_hills` , setting these to zero didn't quite define what most would consider a "low stress" bike route. At least not until now!

This summer, I and the Mapzen routing team enhanced Valhalla's bicycle costing system. Not only is overall routing improved, but `use_roads` and `use_hills` are now much more versatile: when both of these parameters are set to 0, Valhalla generates a nice, "low-stress" biking route. As these parameters get closer to 1, it linearly increases to a more "professional" biking route.

Along with this new update, our default bike settings have also changed — `use_roads` and `use_hills` previously defaulted to 0.5, but now they default to 0.25. The default bike (and thus speed) has also been changed from a road bike to a hybrid bike, which is more suited for city riding.

Technical Details

The meat of this update is the added support for low-stress biking. Our original goal was to introduce a new "low-stress bike" costing class derived from the regular bike costing class. However, after doing this, we realized that the algorithms of the new `LowStressBicycleCost` class were actually cleaner and easier to manage than our original bike costing algorithms. This inspried us to rewrite the old bike code to look like the new low-stress bike code. This way, the `use_roads` and `use_hills` parameters acted as variables that scale the costs of edges (i.e. road segments) with particular attributes in a way that provides low-stress biking at one end of the spectrum, and professional road biking on the other.

When originally making the `LowStressBicycleCost` class, we made use of research by **Michael B. Lowry (http://www.webpages.uidaho.edu/%7Emlowry/)** at the University of Idaho, **Peter Furth (http://www.northeastern.edu/peter.furth/)** at Northeastern, and Tracy Hadden-Loh with the **Rails-to-Trails Conservancy (https://www.railstotrails.org/)**. Their paper titled **"Low-Stress Neighborhood Bikeability Assessment to Prioritize Bicycle Infrastructure" (http://amonline.trb.org/trb60693-2016-1.2807374/t022-1.2817530/362-1.2817851/16-1115-1.2817345/16-1115-1.2817857?qr=1)** provides a way to score a road based on how stressful it is to bike on. Here are the variables that are included in the score:

```
stress_factor = accomodation_factor * road_stress
```

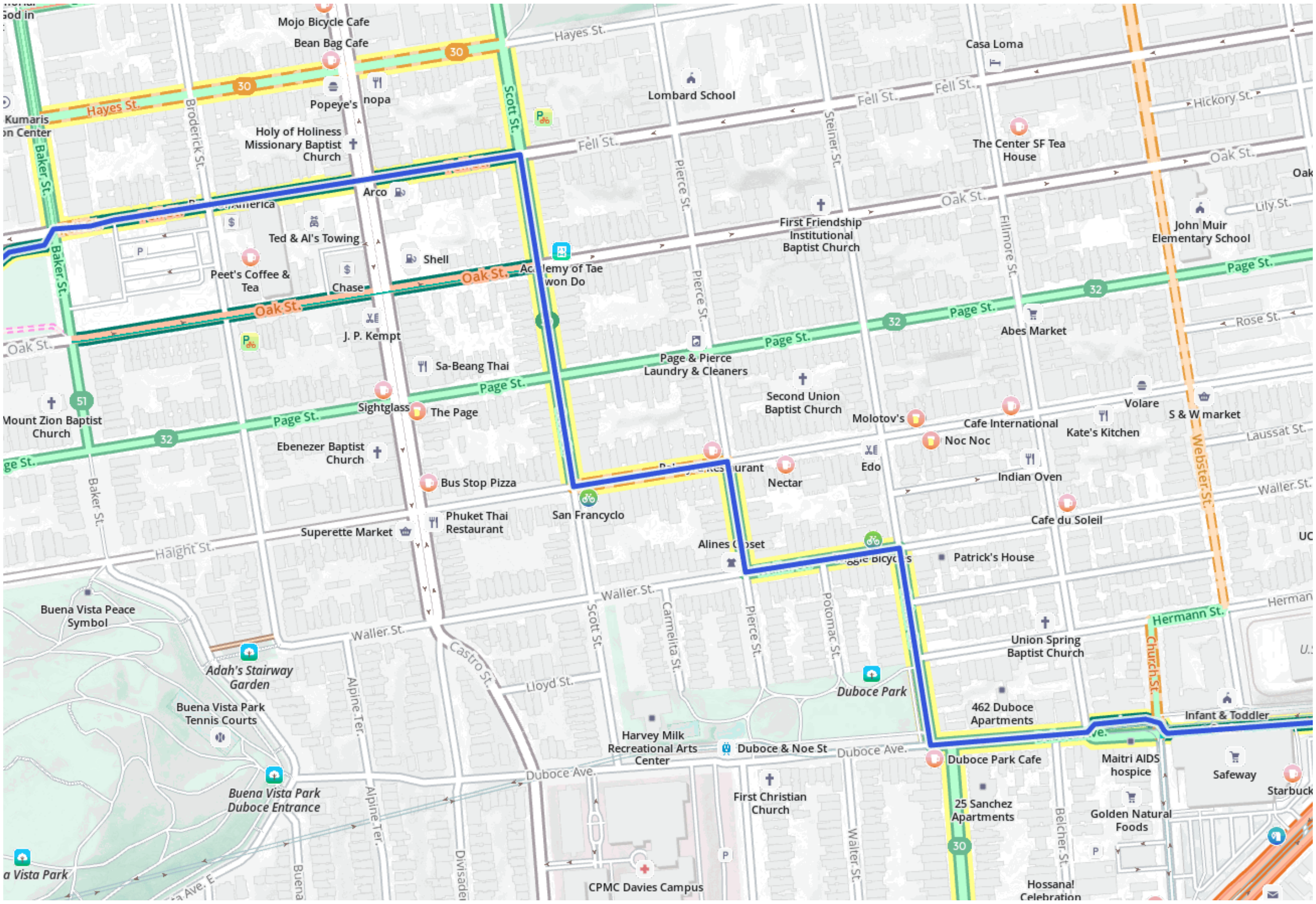
```
edge_weight = 1.0 + stress_factor + slope_factor
```

- `accomodation_factor` is a value between 0 and 1, where 0 means the road is very well accommodated for biking and 1 is not accomodated for biking.
- `road_stress` represents how stressful it is to bike on the road due to things like road speed and lane count. This number can be greater than 1 — the higher it is, the more stressful the road segement.
- Multiplying these two values gets the `total_stress` value which is added with the `slope_factor` and 1 to get `edge_weight` .
- We multiply `edge_weight` at the end of our `EdgeCost` function to the estimated time it takes to travel the edge. This way, edges that have unfavorable attributes for our needs are less likely to be taken.

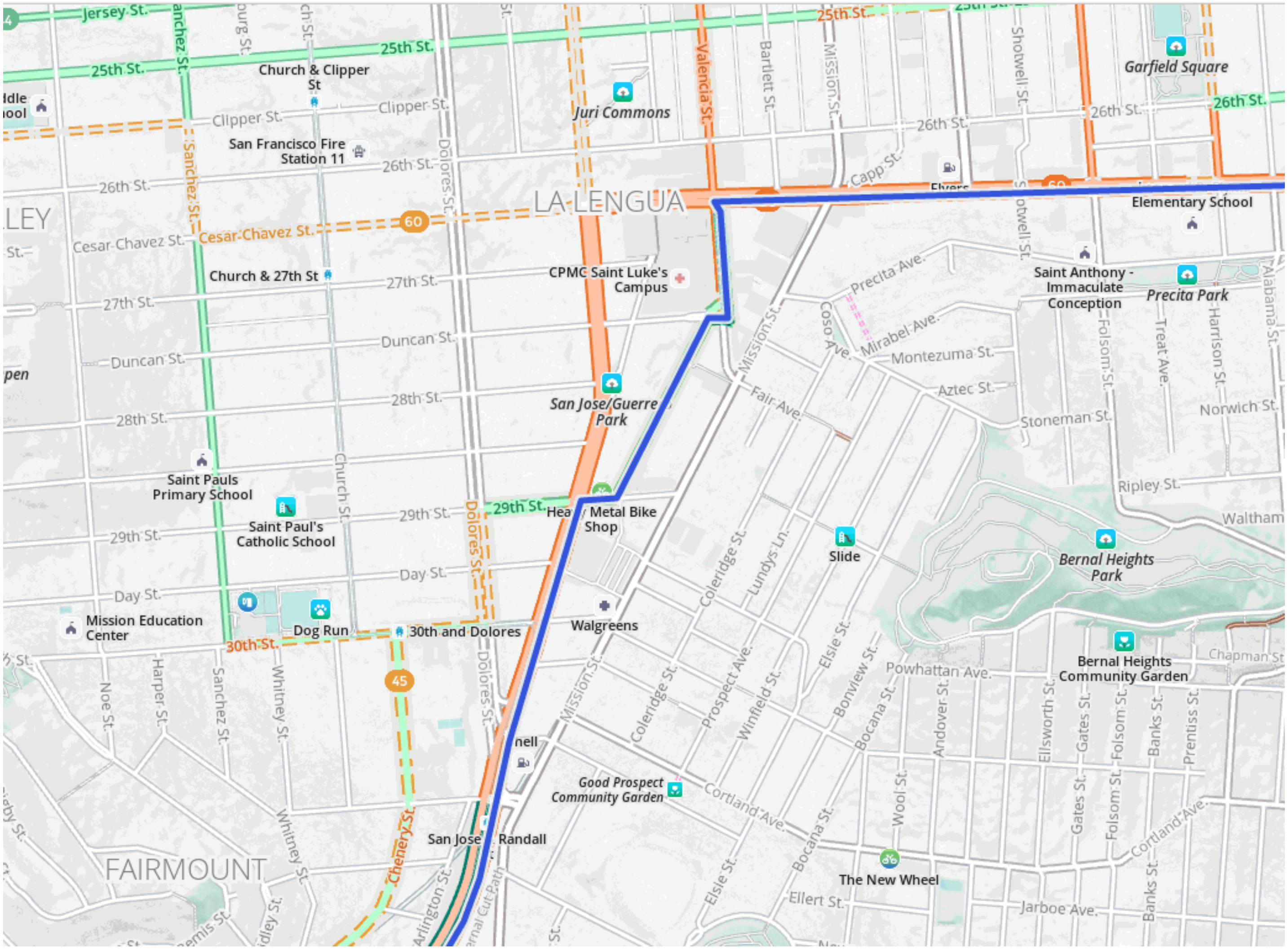
Due to the predictability this algorithm had over our previous one, we folded this code back into the original `BicycleCost` class and removed the `LowStressBicycleCost` class all together, leaving the control with `use_roads` and `use_hills` . As a result, our routing is better and our API has stayed simple and intuitive.

Examples

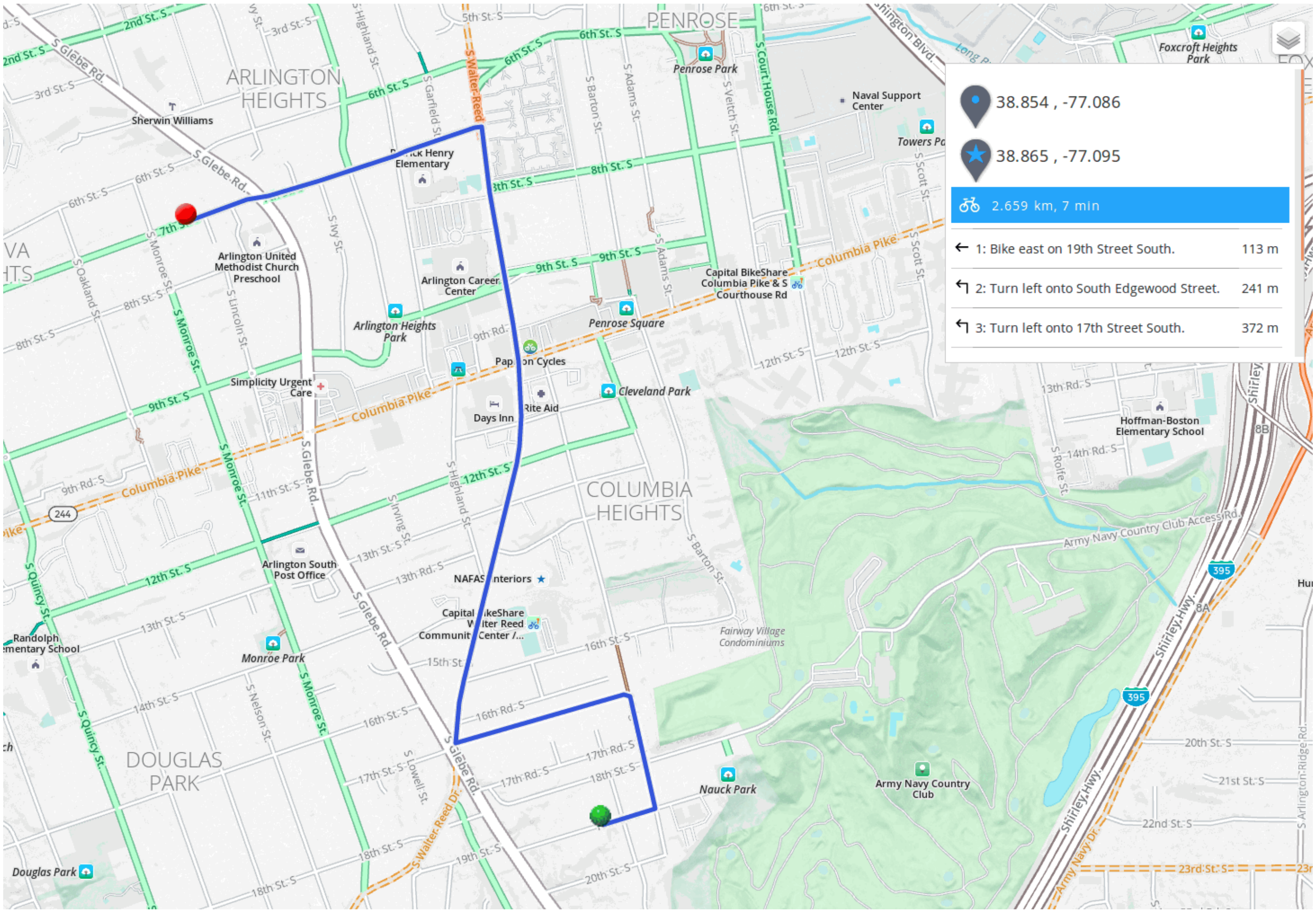
Here are a few examples of the “new and improved” versus the “old” bike routing (both with `use_roads` and `use_hills` set to 0).


Mojo Bicycle Cafe
Bean Bag Cafe
Hayes St.
Popeye's
nopa
Holy of Holiness Missionary Baptist Church
Broderick St.
Scott St.
Lombard School
Casa Loma
Fell St.
Steiner St.
Pierce St.
The Center SF Tea House
Hickory St.
Oak St.
Arco
Ted & Al's Towing
Shell
Peet's Coffee & Tea
Chase
Oak St.
J. P. Kempt
First Friendship Institutional Baptist Church
Fillmore St.
John Muir Elementary School
Lily St.
Page St.
Rose St.
Abes Market
Sa-Beang Thai
Page & Pierce Laundry & Cleaners
Sightglass
The Page
Mount Zion Baptist Church
Ebenezer Baptist Church
Second Union Baptist Church
Molotov's
Cafe International
Noc Noc
Kate's Kitchen
Volare
S & W market
Laussat St.
Edo
Nectar
Indian Oven
Bus Stop Pizza
San Francyclo
Phuket Thai Restaurant
Superette Market
Haight St.
Cafe du Soleil
Waller St.
Alines Closet
Patrick's House
Webster St.
Buena Vista Peace Symbol
Adah's Stairway Garden
Buena Vista Park Tennis Courts
Castro St.
Lloyd St.
Carmelita St.
Potomac St.
Duboce Park
Union Spring Baptist Church
Hermann St.
Church St.
462 Duboce Apartments
Infant & Toddler
Harvey Milk Recreational Arts Center
Duboce & Noe St
Duboce Ave.
Duboce Park Cafe
Maitri AIDS hospice
Safeway
Buena Vista Park Duboce Entrance
Alpine Ter.
First Christian Church
25 Sanchez Apartments
Golden Natural Foods
Belcher St.
Walter St.
CPMC Davies Campus
Hossana! Celebration
Baker St.
30
32
51

Here the old bike route was skipping what is known as “**The Wiggle (https://en.wikipedia.org/wiki/The_Wiggle)**” in San Francisco. The Wiggle is comprised of shared lanes in a zig-zag shape and is the flattest ways to move in that direction. With the low-stress settings, our new and improved bike routing now follows the entire Wiggle.


Jersey St.
25th St.
Church & Clipper St
Clipper St.
San Francisco Fire Station 11
26th St.
Sanchez St.
Cesar Chavez St.
60
Church & 27th St
27th St.
Duncan St.
28th St.
Saint Pauls Primary School
Saint Paul's Catholic School
29th St.
Day St.
Mission Education Center
Dog Run
30th St.
30th and Dolores
45
Chenery St.
San Jose
Randall
FAIRMOUNT
LA LENGUA
Valencia St.
Juri Commons
CPMC Saint Luke's Campus
San Jose/Guerrero Park
29th St.
Heavy Metal Bike Shop
Walgreens
Good Prospect Community Garden
Dolores St.
Church St.
Mission St.
Bartlett St.
Shotwell St.
Capp St.
Garfield Square
26th St.
Elementary School
Saint Anthony - Immaculate Conception
Precita Park
Precita Ave.
Mirabel Ave.
Montezuma St.
Aztec St.
Stoneman St.
Fair Ave.
Coleridge St.
Lundys Ln.
Slide
Prospect Ave.
Winfield St.
Elsie St.
Bonview St.
Bocana St.
Ripley St.
Bernal Heights Park
Bernal Heights Community Garden
Powhattan Ave.
Cortland Ave.
The New Wheel
Ellert St.
Jarboe Ave.
Folsom St.
Gates St.
Banks St.
Prentiss St.
Norwich St.
Waltham
Chapman St
Alabama St.

On this San Francisco route, the old system suggested taking Guerrero Street all the way to the right turn onto Cesar Chavez Street, but the new and improved system takes a shortcut through Tiffany Avenue, then onto the protected Valencia cycleway. While Guerrero has painted bike lanes, it is a very busy street with significant car traffic, while Tiffany Avenue is a **numbered bike route (https://nacto.org/case-study/bicycle-route-and-sign-system-in-san-francisco-ca/)**, on a residential road classified as a **“living street” (http://wiki.openstreetmap.org/wiki/Tag:highway%3Dliving_street)**. The new bike routing gives living streets and numbered routes much higher preference when calculating lower-stress routes.



38.854 , -77.086

38.865 , -77.095

2.659 km, 7 min

1: Bike east on 19th Street South.	113 m
2: Turn left onto South Edgewood Street.	241 m
3: Turn left onto 17th Street South.	372 m

PENROSE
ARLINGTON HEIGHTS
COLUMBIA HEIGHTS
DOUGLAS PARK
Penrose Park
Naval Support Center
Towers Pa
Foxcroft Heights Park
Sherwin Williams
Arlington United Methodist Church Preschool
Arlington Career Center
Arlington Heights Park
Penrose Square
Capital BikeShare Columbia Pike & S Courthouse Rd
Columbia Pike
Papillon Cycles
Simplicity Urgent Care
Days Inn
Rite Aid
Cleveland Park
Hoffman-Boston Elementary School
Arlington South Post Office
NAFAS Interiors
Capital BikeShare Walter Reed Community Center
Monroe Park
Randolph Elementary School
Fairway Village Condominiums
Army Navy Country Club
Army Navy Country Club Access Rd
Nauck Park
Douglas Park
Shirley Hwy
Army Navy Dr
S Glebe Rd
S Walter Reed Dr
S Monroe St
S Court House Rd
244
395

In Arlington, Virginia, the old bike system here suggestes an odd zig-zag route to try and avoid busy streets like South Glebe Road. The new and improved biking system goes straight north, using a bike path and some residential roads to riders' advantage. This is not only quicker but is also safer to ride.

Turn Costing

Turns are tricky business. In Valhalla, turns are penalized to a certain degree to keep routes from becoming ridiculous. However, with low-stress biking, there were certain areas (like the Wiggle and Tiffany Avenue) where we *wanted* more turns. In those situations, it's worth taking a few more turns if it means you can ride along a nice dedicated cycle lanes or a calm side street.

So inside the transition costing, we had to penalize the turns onto low-stress roads a little *less* to allow *more* turns onto those roads. Still, we didn't want to encourage too many unnecessary turns. After some tuning, we think we've struck a reasonable balance for turn costing.

(There aren’t similar issues with `use_hills`, since grade and elevation are calculated along edges, rather than at turning/transition nodes.)

Learn More

You can access the new and improved bike routes, as well as the “low-stress” options now in our **Turn-by-Turn API (https://mapzen.com/documentation/mobility/turn-by-turn/api-reference/)** and **mapzen.js (https://mapzen.com/documentation/mapzen-js/)**!

Or directly make an API request like this:

```
https://valhalla.mapzen.com/route?json= {"costing":"bicycle","costing_options":{"bicycle"
```

In **mapzen.js (https://mapzen.com/documentation/mapzen-js/)**, just tell the built-in router to use `bicycle` costing, and set `use_roads` and `use_hills` to `0` if you want the lowest-stress route.

```
            router = L.Mapzen.routing.control({
                waypoints: [
                    L.latLng(sf_start),
                    L.latLng(sf_stop)
                ],
                router: L.Mapzen.routing.router('your-mapzen-api-key', {
                    costing: "bicyle",
                    costing_options: {
                        "bicycle": {
                            use_roads: 0,
                            use_hills: 0
                        }
                    },
                    directions_options: {
                        language: 'en-US',
                        units: 'miles'
                    }
                }),
            }).addTo(map);
```

Want to display your stress-free bike routes on an attractive map? Or want to browse bike lanes and paths, before you plan a particular trip? See Mapzen's **updated bike map (/blog/bike-map-v2/)**.

· 23 August 2017 ·

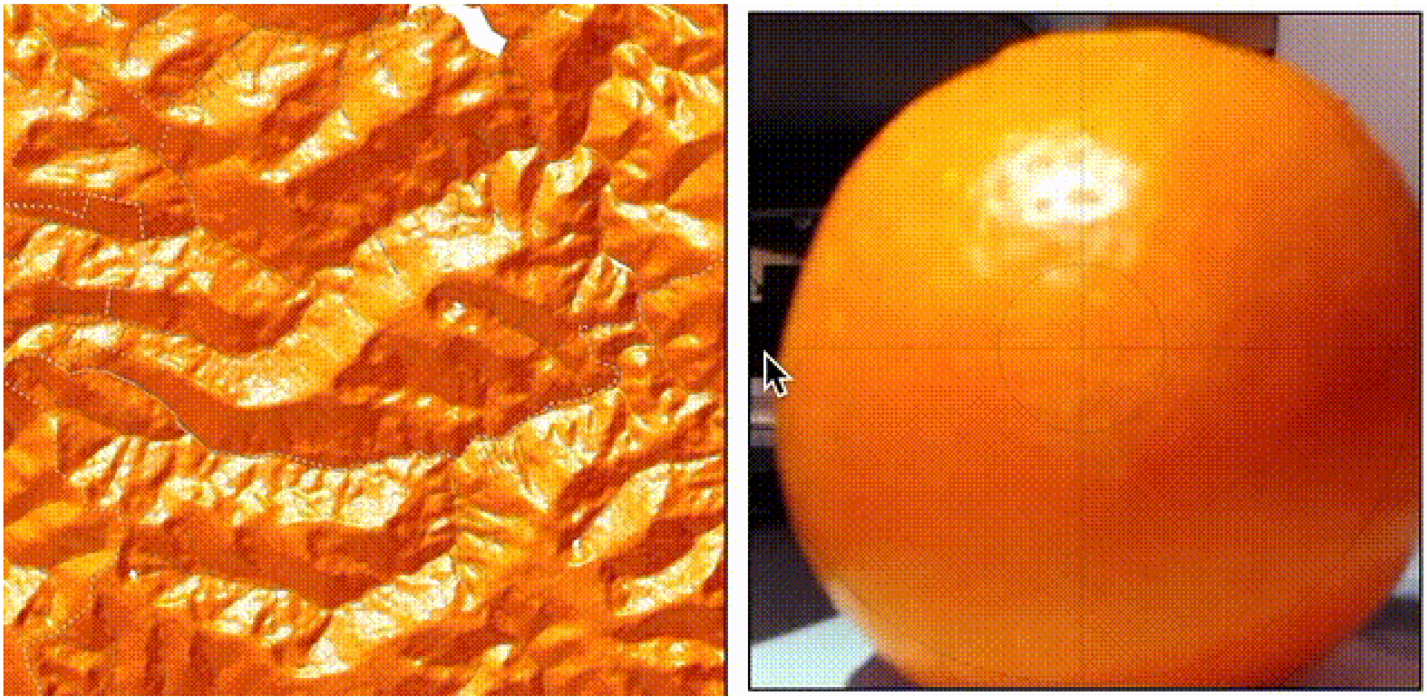
Alex Wilton

Alex is a summer 2017 intern on Mapzen's routing team and is a Computer Science undergrad at Millersville University in Lancaster, PA. When he is not writing sick c++ programs he is most likely writing sick music.

© 2017 Mapzen

Painting Terrain with Kinkade

Here's a gif of me painting some mountains with an orange:



(/projects/tangram/)

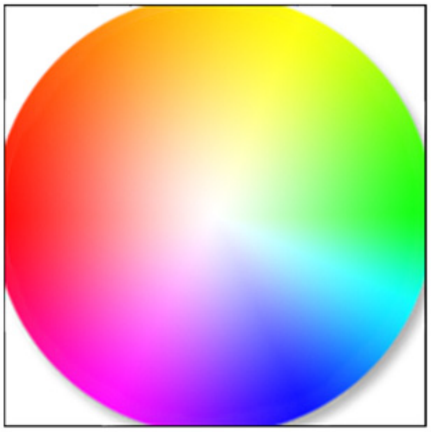
Put another way, I'm shining a light on some open-source terrain data through a webcam, with an open-source mapping engine I work on called **Tangram (https://mapzen.com/products/tangram)**, using a web-based spheremap-painting tool called **Kinkade (https://mapzen.com/tangram/kinkade)**.

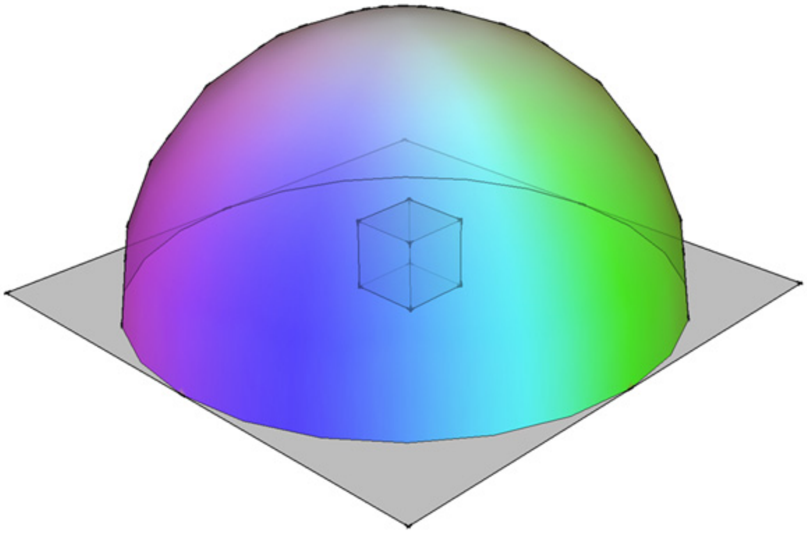
Pestering fruit with flashlights may not sound related to mapping, but when you work with mapping software for a while, everything starts to look like a map. And like a healthy relationship, a map ain't nothing but a way to connect two otherwise unrelated ideas, and I think it's often a useful way to think about things.

In this case, the video of the orange is a kind of map, and the lighting model in the computer is also a kind of map, and the two maps happen to be similar enough that they can have an interesting conversation, which suggests all kinds of other mappy things.

Let me explain.

Sphere We Go Again





The terrain-lighting technique here uses something **I've described previously (https://mapzen.com/blog/mapping-mountains/)** called a spherical environment map (or **spheremap (https://github.com/tangrams/tangram-docs/blob/gh-pages/pages/Materials-Overview.md#mapping-spheremap)**) which colors the faces of a 3D model based on the colors of an image. It's called a spheremap because you can imagine that the model is floating inside a sphere, over which the image is stretched (here we only need half a sphere, because we only ever see one side of the terrain). So every face on the object is pointing at some part of the sphere, and whatever color is on the sphere at that point is "mapped" to the object at the related point.

This technique uses an equation which, under normal circumstances, "maps" the angles of a 3D object to the parts of an image. We've stored all the information we need about the angles of the terrain in our **normal tiles (https://mapzen.com/documentation/terrain-tiles/formats/)** – no actual 3D objects are necessary. So here, the "map" in "spheremap" is the relationship between the input normal and the location of a pixel in the image.

Happily, this relationship is very simple, because the normal value range (-1..1) happens to be very close to the GLSL texture lookup range (0..1). With a simple transposition: `.5*(_normal.xy + 1.)` we can use the normal value itself as the value of the texture lookup. Sweet!

Okay But The Orange Tho

So how is the image of the orange a map? Well. A photograph of an orange is a mapping of the light reflected from the orange received by each pixel in the camera sensor, in the same way that everything you see is a map of the light that manages to bounce through your cornea onto your retina (plus some noise caused by phosphenes).

And from a certain point of view (specifically, **Kant's (https://en.wikipedia.org/wiki/Schema_(Kant))**, see also **Gestalt theory (https://en.wikipedia.org/wiki/Gestalt_psychology)**) what we think of as an "orange" is a cognitive mapping of everything we've previously associated with the concept of "oranges" to a given set of properties belonging to any sufficiently "orange-like" object.

However, the map which is most important to us here is the one made possible by the shape of the orange. Spheres in space act as a tidy sample of the available light in their environment, sorted neatly by angle in three dimensions.

In the world of 3D visual effects, there's **an old trick (http://www.andrew-whitehurst.net/howTo2009.html)** which takes advantage of this fact, involving a shiny ball called a "light probe" photographed on the set of a live-action shot. When used as a spheremap, the image of this mirrored sphere shines the light from a physical scene onto 3D objects in a virtual scene, and the result looks like it was in the original shot.

So the orange is a delicious, citrusy light probe. And the webcam trick works because the image of a spherical orange maps well to the "spherical" part of the spheremap equation. When light hits the orange, the equation picks the light up from the appropriate place in the spheremap, and sends it via the lighting system to the terrain data.

This trick works well with any spherical object, including one of my favorite kinds of spherical objects, **dorodango (https://en.wikipedia.org/wiki/Dorodango)**. These globes of hand-polished mud reflect light in ways which produce particularly earthy terrain maps.

Enough Talking, Let's Map

Like to try this yourself? Remember the web interface I mentioned much, much earlier? It can do all this and more – it even includes some example dorodango to try. We've been using it on some recent updates to the **Mapzen basemap styles (https://mapzen.com/documentation/cartography/styles/)**, to enhance the appearance of terrain relief and integrate that data more harmoniously with its context.

Check out the **Kinkade (https://mapzen.com/tangram/kinkade)** spheremap painter, save the results to **your Mapzen account (https://mapzen.com/dashboard/)**, and let us know what you make!

· 24 August 2017 ·

Peter Richardson

Peter vexes vertices, flusters fragments, and pesters pixels on Mapzen's graphics team.

© 2017 Mapzen

Searching the Summer

search (/tag/search) interns (/tag/interns)

Coming back to Mapzen has been incredibly fulfilling. **Last summer I was an intern on the product team (https://mapzen.com/blog/ladders-for-leaders-ii/)**, where I got my first taste of maps and open source. I started as a complete stranger to the command line and I left with Git, React, and Docker in my toolbox.

This summer, I joined **Pelias (https://github.com/pelias/pelias)**, the team behind **Mapzen Search (https://mapzen.com/search)**. Everything I learned last year made the learning curve less steep. Having worked with Node.js before, I had no trouble diving in and setting up a local instance of Pelias, complete with Elasticsearch, Libpostal and the API.

As a newcomer to the project, my fresh pair of eyes helped illuminate points of confusion in the documentation. For instance, I installed a version of Elasticsearch that Pelias does not support and wondered for a while why the schema could not be created. Once someone on the team was able to point me in the right direction, I watched the hundred of thousands records pour into Elasticsearch. From this first task, I uncovered that box on a map with a magnifying glass icon, and revealed five importers, three microservices, and hundreds of Node modules. I was in awe.

The more familiar I became with the Pelias API, the more I realized **how many regional variations the geocoder text parser needed to consider (https://mapzen.com/blog/libpostal-1-0/)**. Depending on the country, the order of house number and street name varies, as do the order of the region and the zipcode. In some countries, such as Mexico, addresses include the neighborhood or settlement to distinguish streets of the same name but in different parts of the city. Some apartment complex buildings share the same address.

My second project was to create **acceptance tests for international addresses (https://github.com/pelias/acceptance-tests/blob/master/test_cases/international.json)**. I drew examples from a list of US Embassy around the world and compared my expectations to the API results. Failing tests were valuable because they mark and push for better coverage. For instance, one class of issues is alternative names. We would love to search for “Serrano Street, Madrid” and get same point as “Calle de Serrano, Madrid”. However, right now we can only find street names in one language. Prior to joining Pelias, I took all this complexity for granted in a geocoder.

A new part of my workflow was formal code review. **Every pull request (https://github.com/pelias/acceptance-tests/pulls?q=is%3Apr+author%3Atigerlily-he)** was blocked from merging until someone on the team approved the changes. I liked this process quite a bit because I got frequent feedback on my code. In school, the feedback cycle is slow and there is no chance for revision. A code assignment could take two weeks to be graded. By then, I only have vague memories of what I did. Code reviews were also places where discussions sprouted and neatly documented. I could see what everyone else was working on and keep up to date with all the changes in the codebase. For an intern like me, every pull request I didn’t make was a chance to see how good code is written. My standard for writing unit tests came from looking at existing tests.

While a big part of a tech internship is strengthening programming skills, an equally great component is the social skills of team collaboration. Every day begins with a standup meeting, where each person updates the team on what they are working on. Because the Pelias team is semi-remote, this is an important time to connect and bring up questions that are hard to express in written words. I might share my screen and talk through code that I was stuck on or error logs I struggled to decipher. This practice helped me articulate code into words. Very often, I walk away with a good idea how to resolve my problem. Other times, someone may ask for support trying a new tool that another member has expertise on. During these sessions, I have picked up on techniques in asking and providing guidance, as well as generating productive discussion.

Thank you to Diana, Julian, Peter, and Stephen for taking me into the team. I have enjoyed every minute of being at Mapzen and I cannot quantify how much I've learned and grown.

· 29 August 2017 ·

Lily He

Columbia University undergrad, two-time intern at Mapzen (Product and Search teams) via NYC SYEP Ladders to Leaders.

© 2017 Mapzen

TransitFlow — using Python and Processing to visualize regional transit frequency

mobility (/tag/mobility) transitland (/tag/transitland) interns (/tag/interns)

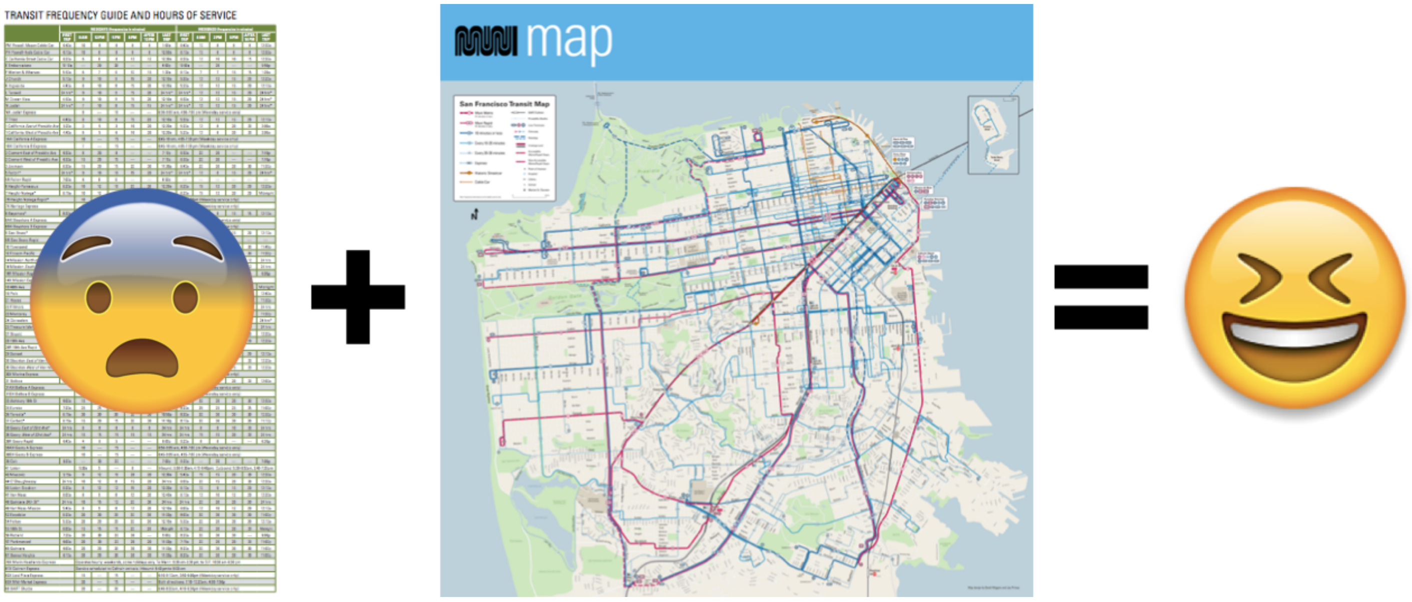
Urban planner **Jarrett Walker (http://humantransit.org/2015/07/mega-explainer-the-ridership-recipe.html)** emphasizes the importance of **transit frequency**:

> *Transit frequency is the elapsed time between consecutive buses (or trains, or ferries) on a line, which determines the maximum waiting time. People who are used to getting around by a private vehicle (car or bike) often underestimate the importance of frequency, because there isn't an equivalent to it in their experience. A private vehicle is ready to go when you are, but transit is not going until it comes. High frequency means transit is coming soon, which means that it approximates the feeling of liberty you have with your private vehicle – that you can go anytime. Frequency is freedom!*

This makes intuitive sense. Transit networks with higher frequency and shorter waiting times will yield a more reliable and empowering experience for passengers than those with lower frequency and longer waiting times.

Analyzing transit frequency in an intuitive way, however, can be difficult. Static transit maps provide geographic context but do not give any information about frequency. Timetables provide information about frequency but can be overwhelming, unintuitive and lacking geographic context. Perhaps we can use spatial-temporal visualization to combine the spatial information of a static transit map with the temporal information of a timetable, and make it easer to think and talk about transit frequency!

This is the motivation behind **TransitFlow (https://github.com/transitland/transitland-processing-animation)**, an experimental set of tools to generate spatial-temporal transit frequency datasets and visualizations from the command line.


TRANSIT FREQUENCY GUIDE AND HOURS OF SERVICE
+
muni map
San Francisco Transit Map
=

Mmmm… timetables. Provided by the San Francisco Municipal Transportation Agency.

TransitFlow consists of two parts:

1. A few python scripts to retrieve and wrangle transit schedule data from **Transitland (https://transit.land/)**, an open-source data service sponsored by **Mapzen (https://mapzen.com/)** that aggregates transit network data from around the world. Transitland enables us to merge transit schedules from different operators without having to worry about the granular details of each operator’s GTFS data.
2. A template to generate a **Processing (https://processing.org/)** sketch to animate that data.

Let’s look at a few examples of what you can do with *TransitFlow*.

Bay Area

-
  ```
  python transitflow.py --name=bay_area --bbox=-123.280334,37.011326,-120.607910,38.955137 --clip_to_bbox
  ```

San Francisco

-
  ```
  python transitflow.py --name=san_francisco --bbox=-122.515411,37.710714,-122.349243,37.853983 --clip_to_bbox
  ```

Chicago

- `python transitflow.py --name=chicago --bbox=-87.992249,41.605175,-87.302856,42.126747 --clip_to_bbox`

Mexico City

- `python transitflow.py --name=mexico_city --bbox=-100.041504,18.453557,-97.701416,20.040450 --clip_to_bbox --date=2016-06-01`

Many more examples available **here (https://vimeopro.com/willgeary/transit-flows)**.

How it works

Getting the data

TransitFlow makes use of three **Transitland API (https://transit.land/documentation/datastore/api-endpoints.html)** endpoints to get the data:

1. **Stops** to get transit stop locations
2. **Routes** to get operator vehicle types
3. **ScheduleStopPairs** to get origin -> destination schedule stop pairs

The `ScheduleStopPairs` endpoint does the bulk of the work. Each `ScheduleStopPair` is an edge between an origin stop and a destination stop. Each `ScheduleStopPair` includes origin departure time and location, destination arrival time and location, and a service calendar which tells you which days a trip is possible. *TransitFlow* searches for all `ScheduleStopPairs` for a specified operator or for many operators within a specified bounding box. It then parses the data, calculates trip durations and bearings, and concatenates everything into a tidy, tabular dataset. It then outputs this dataset to single CSV file, which will drive the animation.

Visualizing the data

TransitFlow generates a **Processing (https://processing.org/)** sketch file based on your command line arguments. The Processing sketch reads in the CSV file from the previous step, uses the **Unfolding Maps (http://unfoldingmaps.org/)** library to load map tiles and convert geolocations into screen positions, and animates vehicle movements from origin stop to destination stop using linear interpolation. (Note that we are animating between stops and not yet following the exact route line.)

How to use it

First, navigate to the **github repository (https://github.com/transitland/transitland-processing-animation)** and follow the relatively painless set-up instructions.

You can visualize a single transit operator by passing in the operator's Onestop ID as a command line argument. What's a Onestop ID, you ask? As part of Transitland's **Onestop ID Scheme (https://transit.land/documentation/onestop-id-scheme/)**, every transit operator, route, feed and stop are assigned a unique identifier called a Onestop ID. You may look up an operator's Onestop ID using the **Transitland Feed Registry (https://transit.land/feed-registry/)**.

For example, the onestop ID for San Francisco BART is `o-9q9-bart`. To visualize BART's schedule on today's date: `python transitflow.py --name=bart --operator=o-9q9-bart`

You can also visualize many operators within a region by passing in your bounding box of interest as a command line argument. I like using **bboxfinder (http://bboxfinder.com/)** to draw a bounding box. The bounding box must be in the format: West, South, East, North. For example, to visualize Portland transit flows on today's date: `python transitflow.py --name=portland --bbox=-123.035889,45.257489,-122.250366,45.798170 --clip_to_bbox`

To view your transit visualization, navigate to `sketches\{name}\{date}\sketch` and open the `sketch.pde` file. This should open the Processing application. Simply click the play button or `command + r` to run the animation.

Command line arguments

Key	Status	Description	Example
`--name`	required	The name of your project	`--name=boston`
`--date`	optional	Defaults to today's date	`--date=2017-08-25`

Key	Status	Description	Example
`--operator`	optional	Operator onestop_id	`--operator=o-drt-mbta`
`--bbox`	optional	West, South, East, North	`--bbox=-71.4811,42.1135,-70.6709,42.6157`
`--clip_to_bbox`	optional	Clip results to bounding box	`--clip_to_bbox`
`--exclude`	optional	Operators to be excluded	`--exclude=o-9-amtrak`
`--apikey`	optional	Mapzen API key	`--apikey=mapzen-abc1234`

Map with us

If you use this tool, we'd love you hear from you! Share your visualizations by tagging @transitland on Twitter or reaching out to us at **hello@mapzen.com (mailto:hello@mapzen.com)**.

We're thinking about ways to make this better!

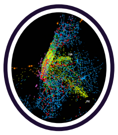
- We would like to add points between stops to make routes with curves and sharp turns smoother at high zooms.
- The animation does not take into account train length or passenger load – if you know of open data sources with this information, let us know!
- We are investigating the best ways to introduce time controls for different start and stop times.
- Real-time data is a longer-term goal for Transitland.

If you have any other suggestions for features, please file an issue in the **Github repo (https://github.com/transitland/transitland-processing-animation)**.

And as a reward for those of you who read all the way to the bottom of this post, here's Bay Area transit set to Claude Debussy's Clair de Lune:

Bay de Lune (https://vimeo.com/231792884) *from* ***Mapzen (https://vimeo.com/user49115026)*** *on* ***Vimeo (https://vimeo.com)****, Clair de Lune by Claude Debussy, via* ***Dreamnote Music (https://audiojungle.net/licenses/terms/music_standard/2.0)***

· 31 August 2017 ·



Will Geary

Will is a summer 2017 intern on Mapzen's Mobility team. A data science grad student at Columbia University, Will finds solace in the fact that he, too, is but a dot skittering around on a map.

© 2017 Mapzen

A "data-driven" approach to improving map-matching, Part II

routing (/tag/routing) mobility (/tag/mobility)

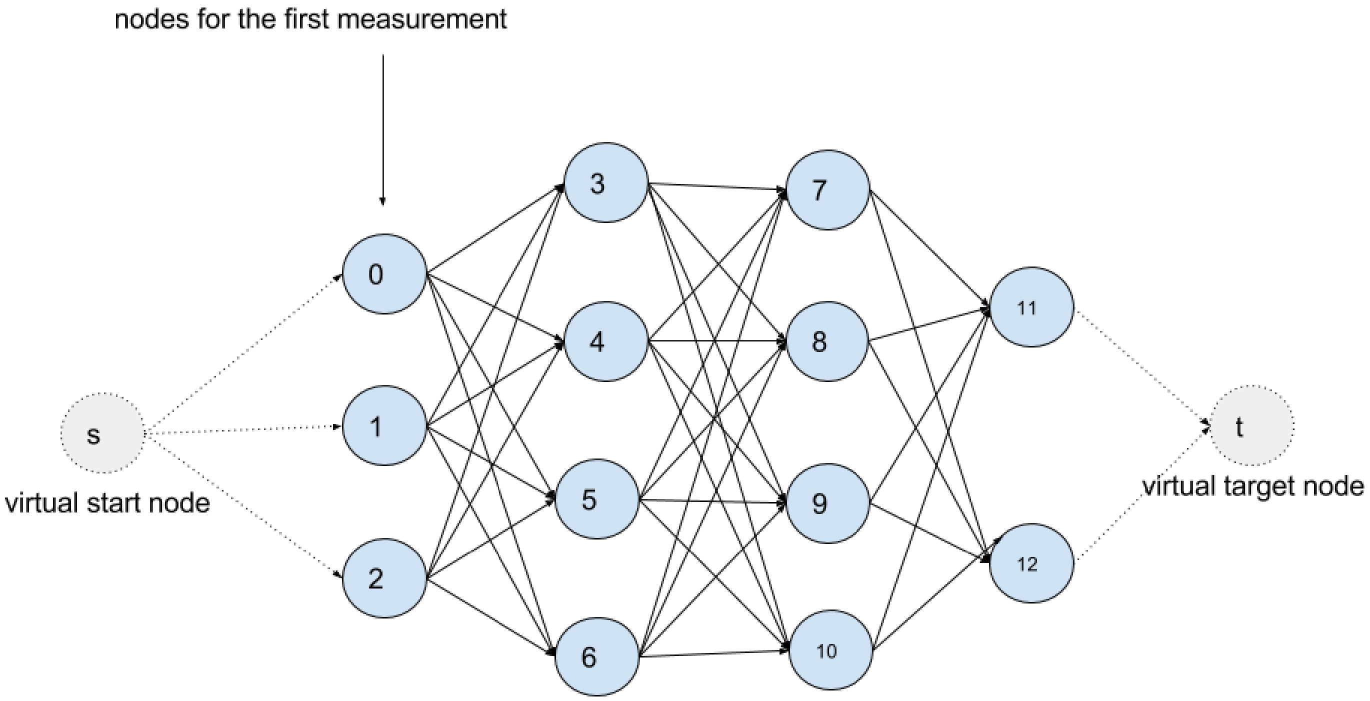
Since we announced our collaboration with the World Bank and more partners to create the Open Traffic platform (/blog/announcing-open-traffic), we've been busy. We've shared two (/blog/open-traffic-osmlr-technical-preview) technical (/blog/osmlr-2nd-technical-preview/) previews of the OSMLR linear referencing system. Now we're sharing more about how we're using Mapzen Map Matching (/blog/map-matching/) to "snap" GPS-derived locations to OSMLR segments, and how we're using a data-driven approach to evaluate and improve the algorithms.

In the **last blog post on Mapzen's map-matching service (https://mapzen.com/blog/map-matching-validation/)**, we showed you how Mapzen uses synthetic GPS data to validate the results of our map-matching algorithm. This time around, we'll dive a bit deeper into the internals of the algorithm itself to see how we can use our validation metrics to fine-tune the map-matching parameters. First, we'll determine parameter values that should work well given any level of GPS noise and frequency. Next, we'll show how to optimize the parameters even further when we know more about the location and characteristics of a GPS data provider.

This post is also available as a **Python notebook (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_II.ipynb)** with code samples!

Parameter Definitions

The Open Traffic Reporter map-matching service is based on the Hidden Markov Model (HMM) design of **Newson and Krumm (2009) (http://research.microsoft.com/en-us/um/people/jckrumm/Publications%202009/map%20matching%20ACM%20GIS%20camera%20ready.pdf)**. HMM's define a class of models that utilize a directed graph structure to represent the probability of observing a particular sequence of events—or in our case, a particular sequence of road segments that defines a route. For our purposes, it is enough to know that HMM's are, in general, governed by two probability distributions which we've parameterized using σ_z and β, respectively. If, however, you are interested in a more detailed explanation of how HMM's work in the context of map-matching, please see the **excellent map-matching primer written by Mapzen's own routing experts (https://github.com/valhalla/meili/blob/master/docs/meili/algorithms.md)**.



The graph structure of an HMM for an example route, from ***Mapzen's map-matching algorithms primer (https://github.com/valhalla/meili/blob/master/docs/meili/algorithms.md)***

The parameter σ_z appears in the equation below which expresses the likelihood of recording a GPS measurement z_t from road segment r_i as the following Gaussian probability density function (PDF):

$$p(z_t|r_i) = \frac{1}{\sqrt{2\pi\sigma_z}} e^{-0.5(\frac{||z_t - x_{t,i}||_{\text{great circle}}}{\sigma_z})^2}$$

where $x_{t,i}$ is the point on road segment r_i nearest the measurement z_t at time t.

In practice, σ_z can be thought of as an estimate of the standard deviation of GPS noise.

The more we trust our measurements, the lower the value of our σ_z should be.

Newson and Krumm (2009) derive σ_z from the median absolute deviation over their dataset, arriving at a value of 4.07, which we have adopted as our default parameter value.

The β parameter comes from the following exponential PDF:

$$p(d_t) = \frac{1}{\beta} e^{\frac{-d_t}{\beta}}$$

where d_t is the difference between the great circle distance and route-traveled distance between time *t* and *t+1*. We use an exponential PDF to define this probability because Newson and Krumm (2009) found that for two successive GPS measurements, the differences in length between the “great circle” distance (i.e. “as-the-crow-flies” distances) and the corresponding route-traveled distance are exponentially distributed:

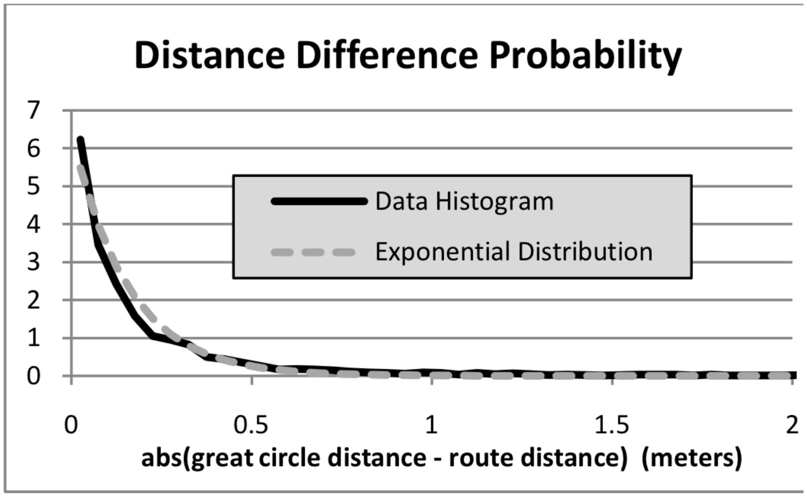


Figure 5: The histogram of $\left| \|z_t - z_{t+1}\|_{great\ circle} - \|x_{t,i} - x_{t+1,j}\|_{route} \right|$ follows an exponential probability distribution.

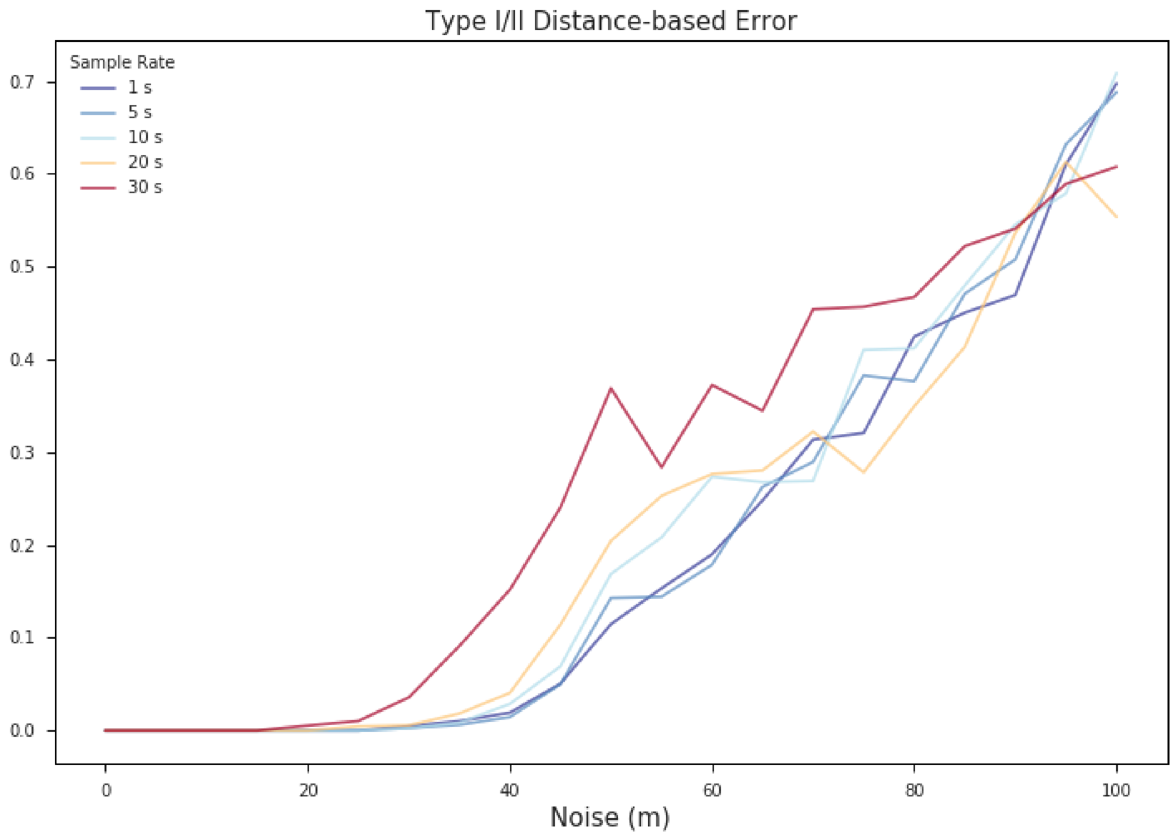
In this context, β can be thought of as the expected difference between great circle distances and route distance traveled, or in other words, the degree of circuitousness of our expected routes.

We have adopted a β of 3 as our default parameter value.

Grid search

Although we can empirically derive our parameter values using the equations above, we can also just search the space of feasible parameter values to find the ones that give the best results. This is a common machine learning approach for algorithm optimization, typically referred to as a *grid search* or *parameter sweep*. Grid search requires a reliable scoring mechanism in order to compare results. Lucky for us, we already implemented a number of these in our exploration of **validation (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/33868a9703f9c00d42dd520339eac3ec04fb4ea0/notebooks/map_matching_part_I.ipynb)** metrics.

Since we have to pick one to use in the grid search, we'll stick with Newson and Krumm's distance-based error metric for now, since it appears most frequently in the literature. Let's take a quick look back at what those errors looked like over our set of validation data:



Clearly the quality of our data, by way of GPS noise and sample rate, has a huge impact on match error. Taking that into consideration, in addition to searching over the feasible space of β's and σ_z's, we'll want to iterate over different noise levels and sample rates so that we can optimize our map-matching regardless of the quality of the data we receive.

Define the search space

Because this grid search takes place in such high dimensional space

$$noise\ levels \times sample\ rates \times \beta\ values \times \sigma\ values \times routes$$

we use a smaller number of discrete noise levels for this section of the analysis:

In [1]:

```
noiseLevels = np.linspace(0, 100, 6)   # in meters
sampleRates = [1, 5, 10, 20, 30]   # in seconds
betas = np.linspace(0.5, 10, 20)   # no units
sigmas = np.linspace(0.5, 10, 20)    # no units
```

Search and score

We've wrapped the entire grid search code into the `grid_search_hmm_params()` function since the internals are pretty messy. Behind the scenes, though, `grid_search_hmm_params()` proceeds as follows:

```
for route in routes:
    for beta in betas:
        for sigma in sigmas:
            for noise in noiseLevels:
                for rate in sampleRates:
                    i) simulate gps
                    ii) match routes
                    iii) score the route
    write scores to a route-specific file
```

Depending on the dimensions of your search space, grid search can be a long and arduous process. In our case, on a 2015 MacBook Pro, it seems to take ~1 min. per route for a given noise level and sample rate. With the configuration specified above, that works out to roughly 30 minutes per route. While this runtime does indeed seem a bit absurd, the beauty of grid search is that if you properly define your search space, you should really only need to do it once.

In [2]:

```
val.grid_search_hmm_params(
    cityName, routeList, sampleRates, noiseLevels, betas, sigmas)
```

Get optimal parameters

Combine the scores for all routes

Because of the slow runtime, `grid_search_hmm_params()` saves its results incrementally in route-specific files. We must first recombine them into a single master dataframe so that we can query, plot, and otherwise manipulate the data all together:

In [3]:

```
frame = val.mergeCsvsToDataFrame(
    dataDir='../data/sf200/', matchString='*.csv')
```

The first five rows of the combined dataframe:

In [4]:

```
frame.head()
```

Out[4]:

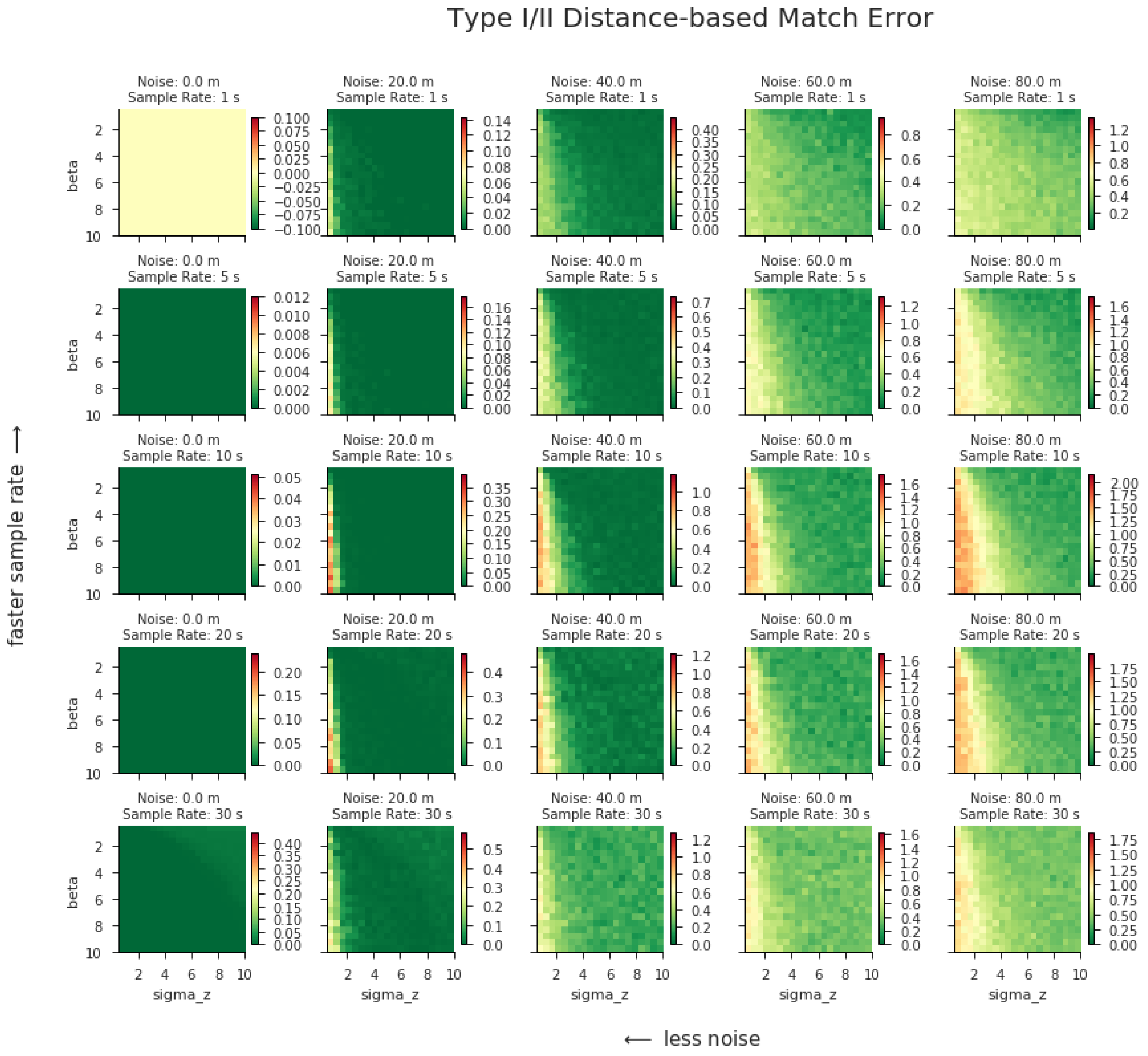
	sample_rate	noise	beta	sigma_z	score
0	1	0.0	0.5	0.5	0.000000
1	5	0.0	0.5	0.5	0.000000
2	10	0.0	0.5	0.5	0.000000
3	20	0.0	0.5	0.5	0.219928
4	30	0.0	0.5	0.5	0.219928

Visualize scores

Now we visually can assess the parameter scores to check for global optima or tease out any other patterns that might persist across the various dimensions of our grid search. The `get_param_scores()` function will also return a dictionary of optimal parameter values corresponding to each one of the subplots below:

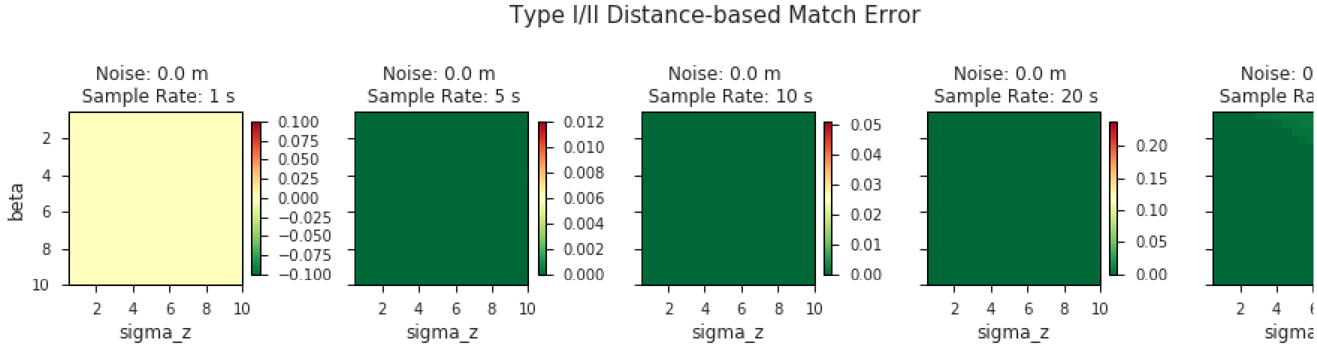
In [5]:

```
paramDict = val.get_param_scores(
    frame, sampleRates, noiseLevels, betas, sigmas, plot=True)
```



In our case there does appear to be a “global” trend of decreasing error toward the upper right quadrants of our plots, which correspond to higher σ_z’s and lower β’s. If we wanted to optimize our parameters across all sample rates and noise levels at once, the results suggest that our optimal parameters would lie at the maximum σ_z and minimum β values of our search space. This is an interesting finding, and not entirely satistfying, since it is typically unwise to pick parameter values at the limits of your search space—who knows what may lie on the other side? Although this could be evidence that we didn’t cast a wide enough net in our parameter sweep, there is reason to believe that it might not be worthwhile to search any further:

1. Focusing specifically at the plots in the 0 - 40 m range of noise, it’s pretty clear that expanding the range of feasible parameter values wouldn’t do any good. **You can’t get much better than 0 error, and our plots in this range are already dominated by dark green. We can obtain optimal results by picking parameters anywhere in this dark green zone.**



Behold small multiples: while the full grid above shows the sample rate along the y-axis, here we've transposed the data with sample rate running along the x-axis and we collapsed the other axis into a GIF.

1. We expect most of our data to be within this bottom half of the range of noise levels we tested. According to the **Institute of Navigation (https://www.ion.org/publications/abstract.cfm?articleID=13079)**, GPS-enabled smartphones can typically achieve an accuracy of ~5 m. Even taking into consideration the fact that most Open Traffic data providers will be operating in dense urban environments where GPS is notoriously inaccurate, we still expect to be operating primarily in the realm sub-60 m of noise. **It is likely not to our advantage to tune our algorithm parameters to optimize for the worst-case scenario.**
2. In the range of noiser data (> 60 m), although there is very little dark green in the plots, there is also no evidence to suggest that we should expect to do any better by increasing our search space. In fact, our inuition about the limitations of map-matching with extremely noisy GPS data tells us that the opposite is probably true. **There is only so much we can expect the algorithm to do with data once its positional accuracy is less than the length of a standard city block.**
3. Performing a grid search over this many different parameters is **extremely slow**.

Taking these ideas into consideration, it is clear that expanding our search space would yield diminishing returns. Instead, we'll use the results we've already got to make a more informed decision about how we want to fine-tune our parameters. In the next section, we'll implement such an approach for one hypothetical transportation network company (TNC).

Case study: “A-OK Rides”

Consider a hypothetical TNC called *A-OK Rides*. A-OK Rides happens to be based in London, and nearly 90% of A-OK trips either start or end in the city's central business district. In order to conserve bandwidth and processing power, the A-OK app only collects GPS measurements from A-OK drivers every 20 seconds. In this case study, we'll see how to use this contextual information about our partner's service territory and anticipated data quality in order to improve our match results.

In [6]:

```
cityName = 'London'
```

In [7]:

```
routes = val.get_POI_routes_by_length(
     cityName, minRouteLength=1, maxRouteLength=5, numRoutes=100, apiKey=gmapsKey)
```

Since we know that almost 90% of our GPS data will be generated from within an area dominated by skyscrapers and other forms of dense urban development, this is a case where we actually *do* want to optimize for noisier data. As such, we will perform our parameter sweep over routes generated with 60 m of noise:

In [8]:

```
noiseLevels = [60]    # meters
```

Similarly, since A-OK only collects data every 20 seconds, we can make sure we are only optimizing for data generated at this frequency:

In [9]:

```
sampleRates = [20]    # seconds
```

We'll search for our optimal β and σ_z values over the same space as before:

In [10]:

```
betas = np.linspace(0.5, 10, 20)
sigmas = np.linspace(0.5, 10, 20)
```

In [11]:

```
val.grid_search_hmm_params(
    cityName, routes[24:], sampleRates, noiseLevels, betas, sigmas)
```

And retrieve the parameter values that gave us the generated the lowest matching error:

In [12]:

```
paramDf = val.mergeCsvsToDataFrame(
    dataDir='../data/', matchString="London*.csv")
```

In [13]:

```
paramDict = val.get_param_scores(
    paramDf, sampleRates, noiseLevels, betas, sigmas, plot=False)
beta, sigma = val.get_optimal_params(
    paramDict, sampleRateStr='20', noiseLevelStr='60')
print('Sigma: {0} // Beta: {1}'.format(sigma, beta))
```

```
Sigma: 10.0 // Beta: 5.5
```

We can now generate new routes, score them using out optimal parameter values, and compare the match errors to what we would have obtained using our default parameters. This will tell us how much our parameter optimization has improved the performance of our map-matching algorithm.

In [14]:

```
newRoutes = val.get_POI_routes_by_length(
    cityName, 1, 5, 100, gmapsKey)
```

In [15]:

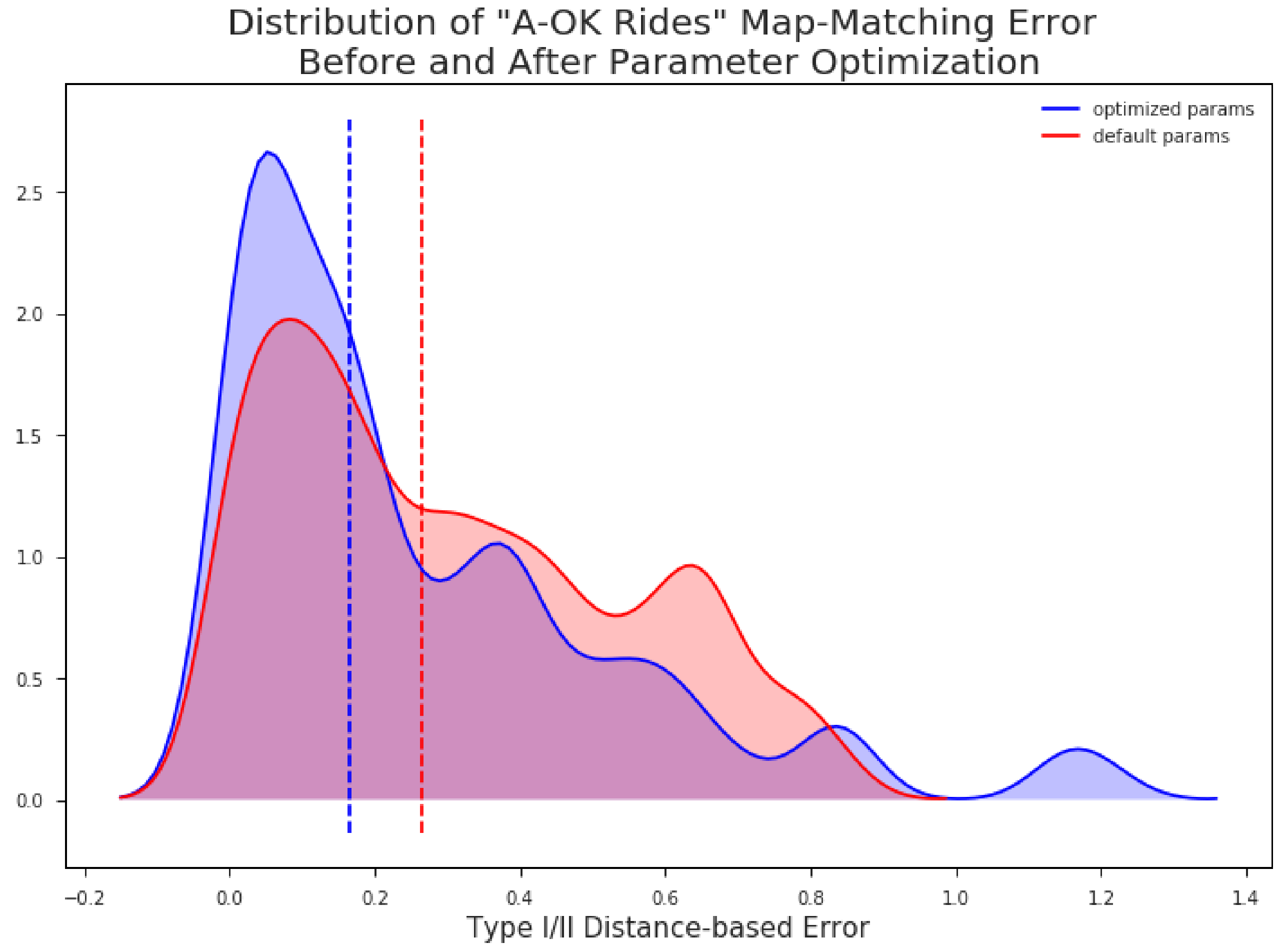
```
tunedDf, _, _ = val.get_route_metrics(
    cityName, newRoutes, sampleRates, noiseLevels,sigmaZ=sigma, beta=beta, saveResults=False)
```

In [16]:

```
defaultDf, _, _ = val.get_route_metrics(cityName, newRoutes, sampleRates, noiseLevels,
saveResults=False)
```

In [17]:

```
val.plot_err_before_after_optimization(defaultDf, tunedDf)
```



By tuning our map-matching parameters to fit the specific character of A-OK Rides' data, we were able to reduce our median error rate from 26% to 17%. Although it might not seem like much on the chart, a **9% gain** in median match accuracy can still have a huge impact on the overall performance of a near-real-time traffic platform like Open Traffic.

Most importantly, however, we've shown that **we have a reliable methodology for fine-tuning our map-matching service to meet the specific needs of our data providers, whoever—and wherever—they may be.**

· 07 September 2017 ·

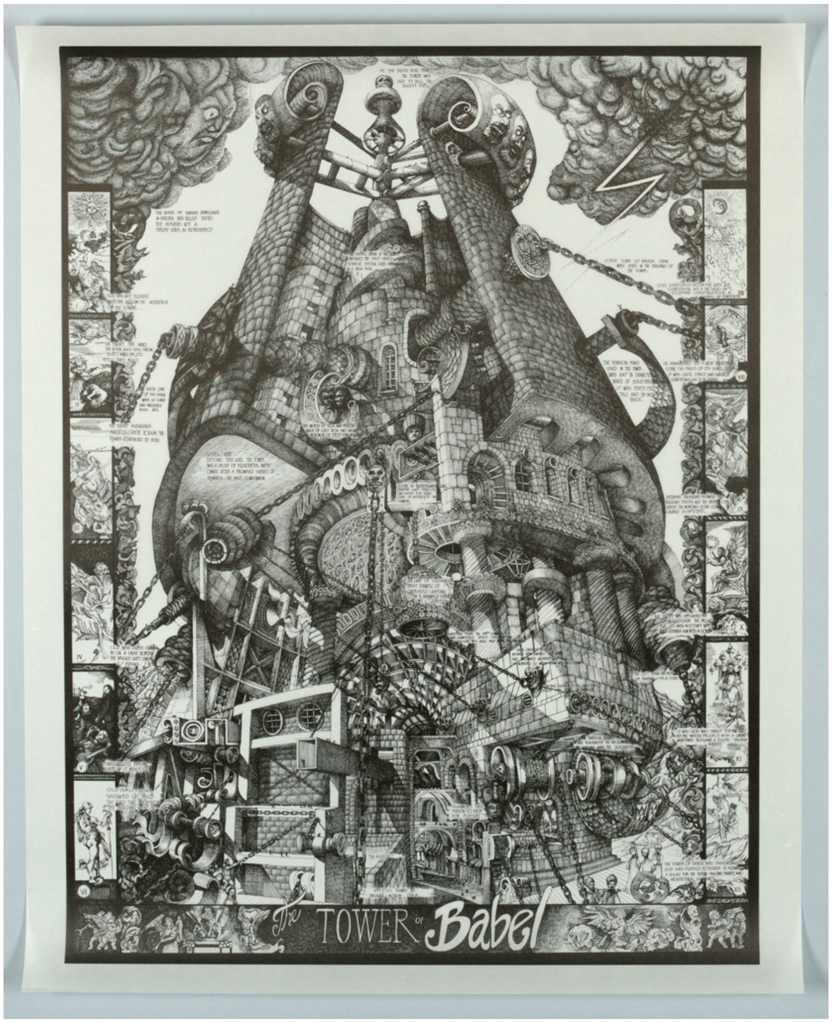
Max Gardner

Max is a summer 2017 intern on the Mapzen Mobility team. A PhD student at UC Berkeley, he will continue to support the Cavs even if LeBron leaves Cleveland again because, after all, Max left too.

© 2017 Mapzen

Geocoding for Polyglots

search (/tag/search) geocoding (/tag/geocoding)



(https://collection.cooperhewitt.org/objects/18642873/)

For a while now Pelias, the engine behind Mapzen Search, has supported searching by the default name of Who's on First records such as **Marseille, France (https://whosonfirst.mapzen.com/spelunker/id/101749199/)**. However, not everyone calls Marseille, France by that name: Our Korean-speaking users may only know it as **마르세유, 프랑스** whereas our Bulgarian-speaking users may refer to it only as **Марсилия ,Франция**. Those used to the Roman alphabet have a hard time finding places like **9丁目 (https://whosonfirst.mapzen.com/spelunker/id/85912437/)**, a neighbourhood of **Hachiōji, Japan (https://whosonfirst.mapzen.com/spelunker/id/102031773/)**.

To address this, we recently released functionality that supports transliterated names not only in the input query but the output results as well.

Perusing **Who's on First data (https://whosonfirst.mapzen.com/spelunker/)** shows many places with several categories of transliterations for a wide variety of languages. The descriptive categories delineate how places are referred to officially and culturally. For example, **New York, NY (https://whosonfirst.mapzen.com/spelunker/id/85977539/)** has a number of names in English:

- preferred: New York
- variant: New York City, NY City
- colloquial: The Big Apple, The Five Boroughs, and Gotham

("Preferred" and "variant" names are the official monikers for places, whereas "colloquial" names are nicknames that have evolved culturally through various means. And for lack of a better term, "administrative hierarchy" will be used to describe the polygons within which an address exists, such as neighbourhood, locality, region, and country.)

The biggest hurdle in supporting non-default names is storage: there are ~500,000,000 address, street, and venue documents stored in Elasticsearch for Pelias and each has to know the administrative hierarchy it belongs to. Storing the textual names for each language would be both excessive and wildly redundant. There are over **300 transliterated preferred, variant, and colloquial names for France (https://whosonfirst.mapzen.com/spelunker/id/85633147/)** alone in **Who's on First (https://whosonfirst.mapzen.com/)**, in a wide variety of languages. Adding language codes and transliterations to each would significantly add to the index size – if this were done, ***every address*** in France would have the name France with ***all*** France's variants stored. Certainly Elasticsearch has mechanisms to tidy this up a bit internally, but this approach is still wildly duplicative. Additionally, querying Elasticsearch by text is cumbersome but filtering by IDs is far easier and faster.

To account for this, we added an internal service called **Placeholder (https://github.com/pelias/placeholder)** that provides two key functions to Pelias:

- parsing administrative area parts of queries, such as "Luxembourg" and "Lancaster, PA"
- looking up language transliterations

Pelias still relies upon **libpostal (https://github.com/openvenues/libpostal)** as the first parser but since it only returns a single analysis result, there's a chance that it could misinterpret the administrative area parts of the input. That is, "Ontario, CA" can legitimately be interpreted as both **Ontario, Canada (https://whosonfirst.mapzen.com/spelunker/id/85682057/)** and **Ontario, California (https://whosonfirst.mapzen.com/spelunker/id/85923933/)**.

Placeholder specializes in geodisambiguation, giving the API the ability to search for an address in various applicable contexts rather than just one. Continuing with the previous example, Placeholder is used to disambiguate all the things that "Ontario, CA" could be so for inputs like **309 E Sunkist St Ontario, CA (https://mapzen.com/search/explorer/?query=search&text=309%20E%20Sunkist%20St%20Ontario%2C%20CA)** and **35 Big Tub Rd Ontario, CA (https://mapzen.com/search/explorer/?query=search&text=35%20Big%20Tub%20Rd%20Ontario%2C%20CA)**, Pelias can unambiguously find the correct addresses in Ontario, California and Ontario, Canada, respectively, despite libpostal identifying "Ontario, CA" as a city/state in both cases:

```
> 309 E Sunkist St Ontario, CA

Result:

{
  "house_number": "309",
  "road": "e sunkist st",
  "city": "ontario",
  "state": "ca"
}

> 35 Big Tub Rd Ontario, CA

Result:

{
  "house_number": "35",
  "road": "big tub rd",
  "city": "ontario",
  "state": "ca"
}
```

The advantage of this geodisambiguation approach becomes very evident in the case of **603 Euclid Ave Ontario, CA (https://mapzen.com/search/explorer/?query=search&text=603%20Euclid%20Ave%20Ontario%2C%20CA)** as this address exists in both Ontario, California and Toronto, Ontario, Canada, allowing Pelias to return both results.

Additionally, because Placeholder already has the data needed to interpret inputs in a variety of languages, it also has the ability to look up those transliterations for output.

Input

You don’t need to specify language in a query — just enter the address in the language of choice. For example, **Марсилия ,Франция (https://mapzen.com/search/explorer/?query=search&text=%D0%9C%D0%B0%D1%80%D1%81%D0%B8%D0%BB%D0%B8%D1%8F%20%2C%D0%A4%D1%80%D0%B0%D0%BD%D1%86%D0%B8%D1%8F)** will find Marseille, France in Bulgarian whereas **마르세유, 프랑스 (https://mapzen.com/search/explorer/?query=search&text=%EB%A7%88%EB%A5%B4%EC%84%B8%EC%9C%A0%2C%20%ED%94%84%EB%9E%91%EC%8A%A4)** will find it in Korean. (Note that the results are in the records’ default language. The next section will cover controlling the output language.)

The city and country don’t even have to be the same language. Pelias will handle the combination, such as mixing Korean and Bulgarian for the city and country to produce **마르세유, Франция (https://mapzen.com/search/explorer/?query=search&text=%EB%A7%88%EB%A5%B4%EC%84%B8%EC%9C%A0%2C%20%D0%A4%D1%80%D0%B0%D0%BD%D1%86%D0%B8%D1%8F)**.

Various languages also work with addresses included in the query, such as **200 Boulevard Baille, مارسيليا, فرنسا (https://mapzen.com/search/explorer/?query=search&text=200%20Boulevard%20Baille%2C%20%D9%85%D8%A7%D8%B1%D8%B3%D9%8A%D9%84%D9%8A%D8%A7%2C%20%D9%81%D8%B1%D9%86%D8%B3%D8%A7)** in Arabic and **200 Boulevard Baille, マルセイユ, フランス (https://mapzen.com/search/explorer/?query=search&text=200%20Boulevard%20Baille%2C%20%E3%83%9E%E3%83%AB%E3%82%BB%E3%82%A4%E3%83%A6%2C%20%E3%83%95%E3%83%A9%E3%83%B3%E3%82%B9)** in Japanese.

Recognition of transliterated names currently works on the `/search` and `/search/structured` endpoints.

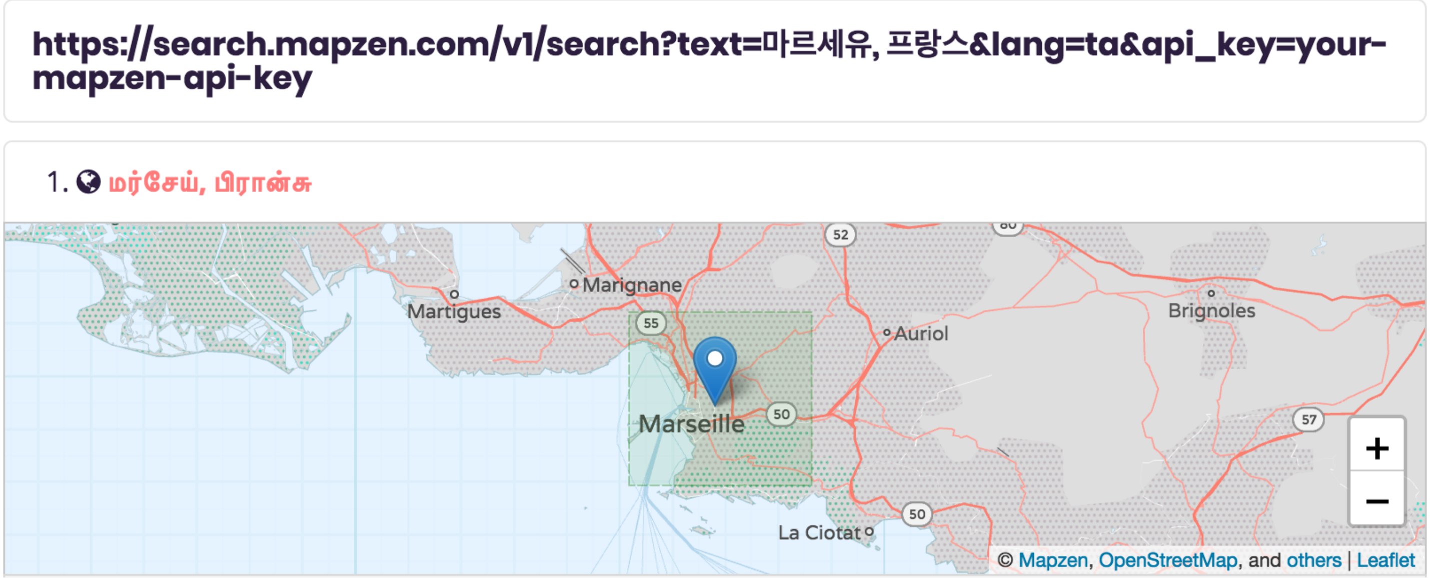
Output

With the release of this functionality, not only does Pelias understand the administrative hierarchy in a variety of languages, the results from a request can be output in a specific language. Doing so can be done by calling the API in one of two ways:

- `lang` parameter
- `accept-language` header

The `lang` parameter is checked first and favored over the `accept-language` header. Pelias currently supports two-character language codes in the **list of ISO 639-1 standard (http://www.loc.gov/standards/iso639-2/php/English_list.php)**. A warning is returned if the requested language is unknown. (Three-character codes are on our to-do list.)

Marseille, France can be returned in **Italian (https://mapzen.com/search/explorer/?query=search&text=Marseille%2C%20France&lang=it)** (Marsiglia, Francia) and **Icelandic (https://mapzen.com/search/explorer/?query=search&text=Marseille%2C%20France&lang=is)** (Marseille, Frakkland). Different language inputs and an entirely different output language work, too. For example, Marseille, France in a combination of Chechen and Arabic can be input and then output in **Tamil (https://mapzen.com/search/explorer/?query=search&text=%D0%9C%D0%B0%D1%80%D1%81%D0%B5%D0%BB%D1%8C%2C%20%D9%81%D8%B1%D9%86%D8%B3%D8%A7&lang=ta)** or **Haitian (https://mapzen.com/search/explorer/?query=search&text=%D0%9C%D0%B0%D1%80%D1%81%D0%B5%D0%BB%D1%8C%2C%20%D9%81%D8%B1%D9%86%D8%B3%D8%A7&lang=ht)**.



https://search.mapzen.com/v1/search?text=마르세유, 프랑스&lang=ta&api_key=your-mapzen-api-key

1. மர்சேய், பிரான்சு

Note that there are not transliterations for every place on earth. In the case where a transliteration is not found for the requested language, Pelias returns the default name for that place, though other parts of the administrative area may be converted. For example, there's no transliteration of Lancaster to Thai so specifying `lang=th` for **Lancaster, PA (https://mapzen.com/search/explorer/?query=search&text=lancaster%2C%20pa&lang=th)** would return "Lancaster" as the locality, "รัฐเพนซิลเวเนีย" as the region, and "สหรัฐอเมริกา" as the country. If you're knowledgeable in a certain language and would like to help out Who's on First with transliterations, please either submit changes via **Boundary Issues (https://whosonfirst.mapzen.com/boundaryissues/)** or create a **GitHub issue (https://github.com/whosonfirst-data/whosonfirst-data/issues)**.

The Glorious Future

Now that we support transliterations in a variety of languages for neighbourhoods, cities, countries, and everything else in the administrative hierarchy, we'll start working on doing the same for address and venue names. Stay tuned!

Image credits:

- **Poster, The Tower of Babel; Gift of Unknown Donor; 1996-128-4 (https://collection.cooperhewitt.org/objects/18652931/)**

· 08 September 2017 ·

Stephen Hess

Stephen works on geocoding exclusively as a means to fund his passion for designing and building wooden frames for old maps.

© 2017 Mapzen

Welcoming Mapzen Places and Libpostal to Mapzen Flex

data (/tag/data) whosonfirst (/tag/whosonfirst) places (/tag/places) flex (/tag/flex)

Earlier this year, **we launched the beta of the Who's on First API (https://mapzen.com/blog/whosonfirst-api/)** enabling users of the Who's On First gazetteer to programmatically query for data.

We've continued to improve the service, and starting October 1, we are renaming the Who's on First API to Mapzen Places and adding it to **Mapzen Flex (https://mapzen.com/pricing/)**, with clear, predictable pricing and a generous free tier.

Mapzen Places is based on open data and can be used with any map or platform, with business-friendly licensing. Get points and polygons with hierarchies and concordances, find neighbourhoods, postal codes, and a growing list of venues. **Language coverage (https://mapzen.com/blog/summer-2017-wof/)** includes solid improvements to placenames around the world in 20 languages, including Dutch, French, Spanish, Portuguese, Russian, Polish, German, Chinese, Vietnamese, Swedish, Arabic, Japanese, Indonesian, and Korean.

We're also bringing **Libpostal (https://mapzen.com/documentation/libpostal/)** to Mapzen Flex! Thanks to machine learning you can parse and expand international address data with **99.45% parsing accuracy (https://mapzen.com/blog/libpostal-1-0/)** in 249 countries and over 60 languages, including East-Asian address formats.

Sign up for a key today (https://mapzen.com/dashboard) and **get ready to Flex (https://mapzen.com/pricing/)**!

image: Mural, Stamp World Map, 2011 Sir Paul Smith via ***Cooper Hewitt (https://collection.cooperhewitt.org/objects/404577581/)***

· 14 September 2017 ·

© 2017 Mapzen

Statoids, Mesoshapes, and Who's On First

whosonfirst (/tag/whosonfirst) data (/tag/data)

(https://mapzen-assets.s3.amazonaws.com/images/statoids-meso/main_photo.jpg)

Recently, we've been working with two datasets, **Mesoshapes**, which we've **previewed before (https://mapzen.com/blog/mesoshapes/)**, and **Statoids**, the premier encyclopedia of administrative subdivisions.

We are happy to announce a partnership with Statoids to allow their data to be included in Who's On First and **Mapzen Places (https://mapzen.com/pricing/)** under generous CC-BY terms. Both these datasets have been merged with existing (and new) Who's On First records and provide nearly complete global coverage at the `region` and `county` placetypes and bring in additional properties like population values and localized placetype names. Please have a read and explore our new additions!

Mesoshapes

We previously **wrote about (https://mapzen.com/blog/mesoshapes/)** our work of increasing county coverage in the **Who's On First (https://whosonfirst.mapzen.com)** gazetteer. This work involved importing new, more detailed geometries for secondary administrative divisions around the globe, as well as importing more place-specific properties for each record. We have since wrapped up the Mesoshapes work and are proud to now maintain just over **46,500** `county` records in Who's On First!

This isn't all we've done; we were also able to update records for primary divisions in each country (`region` records), too. We verified and increased the accuracy of `region` records in Who's On First for each of the 110+ countries delivered by Mesoshapes, adding them to the `county` records already in Who's On First!

Recently, we've joined this work with a new dataset, **Statoids (http://www.statoids.com/)**, to substantially enhance our records with *many* new properties that we would like to outline below.

Statoids

What exactly *is* Statoids?

Statoids is an online database that organizes the primary and secondary administrative divisions for all countries, dependencies, and disputed areas of the world. These divisions have unique identifiers, which we call the `hasc:id` (you can read more about HASC, **here (http://www.statoids.com/ihasc.html)**). The Mesoshapes import included many `hasc:id` values; by linking the HASC concordance values, we imported this encyclopedia of geographic data into Who's On First.

We've long admired Statoids for its global coverage and attention to detail. For over 15 years Gwillim Law was the force behind the printed version of **Statoids (https://www.amazon.com/exec/obidos/ASIN/0786407298/)** followed by copious updates on **Statoids.com (http://www.statoids.com)**. We were saddened to hear of Gwillim's **passing (http://www.statoids.com/mgtletter.html)** August 31, 2016.

In talking to Gwillim's daughter Shirley over the last year we came up with a way to memorialize his work: We are happy to announce a partnership with Statoids to allow their data to be included in Who's On First under generous CC-BY terms. We hope Gwillim's work finds an even larger audience, and together we can continue the work he started. The full terms are in our **sources (https://github.com/whosonfirst/whosonfirst-sources/blob/master/sources/README.md#statoids)** repo.

What does Statoids contain?

The Statoids database has plenty of useful data that Who's On First was able to harvest. This data comes in the form of data tables, separated by parent country and administrative division; take a look at the **Statoids page for secondary divisions in Norway (http://www.statoids.com/yno.html)**, for an example. After crafting various Python scripts to handle the Statoids data tables, we were able to bring in a variety of new properties and concordances to Who's On First. Below is a list of harvested data we were able to pull into Who's On First from Statoids:

Concordances to:

- **Dial codes (https://en.wikipedia.org/wiki/List_of_country_calling_codes)**
- **Distinguishing sign codes (http://www.unece.org/trans/welcome.html)**
- **FIFA codes (http://www.rsssf.com/miscellaneous/fifa-codes.html)**
- **GAUL (Food and Agriculture Organization) (http://www.fao.org/geonetwork/srv/en/metadata.show?id=12691)**
- **International Olympic Committee (https://www.olympic.org/the-ioc)**
- **ISO (https://www.iso.org/home.html)**
- **Identity Radio Transmitter (https://www.itu.int/online/mm/scripts/gensel8)** codes
- **MAchine-Readable Cataloging codes (http://www.loc.gov/marc/countries/cou_home.html)** (MARC) from the Library of Congress
- **World Meteorlogical Organization (http://icoads.noaa.gov/metadata/wmo47/wmo_quarterly/47CodeTables9903.html)**
- Unique identifiers (see our **whosonfirst-sources (https://github.com/whosonfirst/whosonfirst-sources)** repository for more information on the `hasc:id`)

Properties, including:

- Names
- Population counts
- Names of Capital cities and areas
- Historical notes about each place
- Timezones
- Translations
- Area (kilometers and miles, squared)
- A brief overview of each placetype "set" per country
- Dates of the most recent Census figures

- Local terms for the administrative division

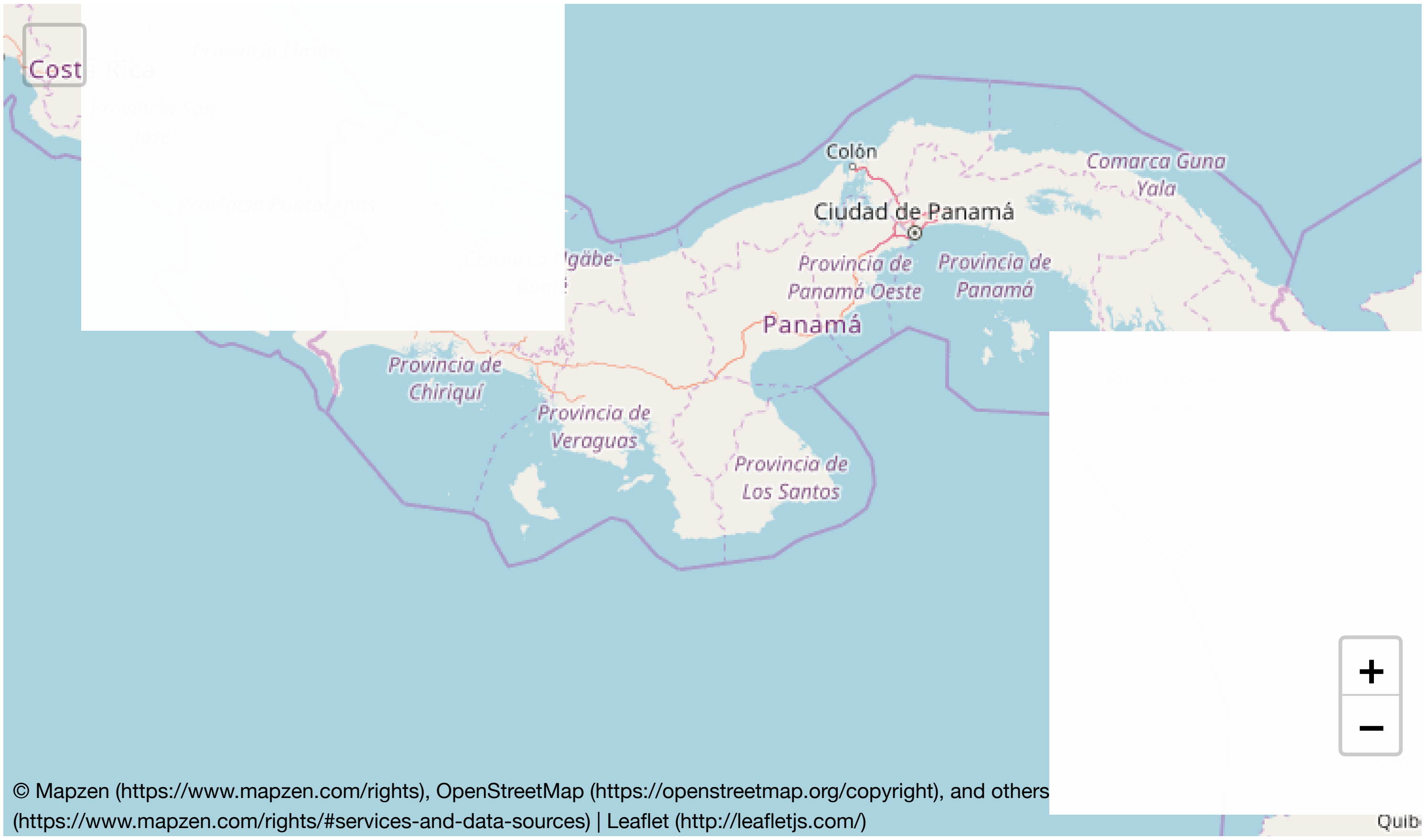
Property Specifics…

We can store each property as a top-level `statoids:` property (see our **whosonfirst-property (https://github.com/whosonfirst/whosonfirst-properties)** repository for detailed information on each `statoids:*` property), harvest new `name:*` properties, link capital cities to their parent(s), link timezone records to the administrative division, and build population and population-related properties from the official population counts in Statoids.

A list of newly harvested Statoids properties, below:

- `src:population`
- `statoids:area_km`
- `statoids:area_mi`
- `statoids:capital`
- `statoids:country`
- `statoids:as_of_date`
- `statoids:hasc`
- `statoids:name`
- `statoids:population`
- `statoids:statoid`
- `statoids:timezone`
- `statoids:type`
- `wof:population`
- `wof:population_rank`
- `src:population`
- `wof:placetype_local`
- various `name:*` properties

Below you can see HASC codes and populations for Panama, Yemen, and the Central African Republic, showing the HASC code and population for both `region` and `county`, along with the localized name for the `county`. Color intensity is relative to the `region` and `county` population data. Note that we're filtering and revealing data and labels at different zoom levels – zoom in to explore, and **open full-screen for variable zoom (/resources/blog/statoids/index.html)**!


Cost Rica
Colón
Comarca Guna Yala
Ciudad de Panamá
Provincia de Panamá Oeste
Provincia de Panamá
Panamá
Provincia de Chiriquí
Provincia de Veraguas
Provincia de Los Santos
+
−
Quib

© Mapzen (https://www.mapzen.com/rights), OpenStreetMap (https://openstreetmap.org/copyright), and others (https://www.mapzen.com/rights/#services-and-data-sources) | Leaflet (http://leafletjs.com/)

Provinces of Panama

Open full-screen (/resources/blog/statoids/index.html) *and* ***check out the scene file in Tangram Play (https://mapzen.com/tangram/play/?api=22/1018#6.167/7.403/20.723)*** *to see how we built it. We aggregated this data using the* ***Who's On First Descender (https://whosonfirst.mapzen.com/spelunker/download/85632499/?exclude=nullisland)****.*

Now, an example…

Let's take a look at the Who's On First record for **Queensland, Australia (https://whosonfirst.mapzen.com/spelunker/id/85681463/)**. Queensland gained both top-level `statoids:*` properties, as well as `wof:*` , `src:*` , and concordance properties.

Here are the new Statoids properties and how they apply to Queensland:

Property Name	Description
`"statoids:country":"AU"`	The country code used by Statoids for the record's parent country (in this case, **Australia (https://whosonfirst.mapzen.com/spelunker/id/85632793/)**

Property Name	Description
`"statoids:gec":"AS04"`	The `gec` identifier for Queensland (also stored in concordances)
`"statoids:hasc":"AU.QL"`	The `hasc:id` for Queensland (also stored in concordances)
`"statoids:iso":"QLD"`	The ISO alpha-3 code for Queensland (also stored in concordances)
`"statoids:name":"Queensland"`	The English, UTF-8 name for this record in Statoids
`"statoids:population":"4332739"`	The catalogued population for Queensland, see below
`"statoids:type":"state"`	What locals call this administrative division, see below
`"statoids:tz":"+10"`	The timezone of Queensland, represented by the difference from GMT
`"wof:placetype_local":"state"`	A top-level `wof:*` property, harvested from the `statoids:type` property
`"wof:population":4332739`	A top-level `wof:*` property, harvested from the `statoids:population` property
`"wof:population_rank":12`	A top-level `wof:*` property, an integer value (0-18), built from `wof:population`
`"src:population":"statoids"`	The source used to build `wof:population`

Results

As of today, Who’s On First has added properties for **256** top-level administrative entities (`country` and `dependency` records). Additionally, **3,811** primary administrative divisions were updated with new properties and concordances and **22,993** secondary administrative divisions were updated with new properties and concordances. Take a look at the new `statoids:*` properties and all the concordances we now have! Links to selected updated records at the `country` , `region` , and `county` placetypes are below.

A few examples, by placetype

Country

A few top-level countries that received new `statoids:*` properties and concordances:

- **Lesotho (https://whosonfirst.mapzen.com/spelunker/id/85632173)**
- **Myanmar (https://whosonfirst.mapzen.com/spelunker/id/85632181)**
- **Qatar (https://whosonfirst.mapzen.com/spelunker/id/85632299)**

Region

Some primary divisions that received new `statoids:*` properties and concordances:

- **South Chungcheong, South Korea (https://whosonfirst.mapzen.com/spelunker/id/85673203/)**
- **Addis Ababa, Ethiopia (https://whosonfirst.mapzen.com/spelunker/id/85671149/)**
- **San José, Costa Rica (https://whosonfirst.mapzen.com/spelunker/id/85670391/)**

County

Examples of secondary divisions that received new `statoids:*` properties and concordances:

- **Tariat, Arhangay, Mongolia (https://whosonfirst.mapzen.com/spelunker/id/1092041727/)**
- **Paktha, Bokeo, Laos (https://whosonfirst.mapzen.com/spelunker/id/1092030459/)**
- **Vlorë, Sarandë, Albania (https://whosonfirst.mapzen.com/spelunker/id/421179955/)**

Next Steps?

This work is far from over. Follow-up issues to untangle recent administrative changes in all countries have been filed, and additional concordance work will be done to improve our records with more top-level properties, more names, more detailed geometries and alt-geometries, and add even more concordances. We are working to import more properties from **GeoNames (https://www.geonames.com)**, **Wikidata (https://www.wikidata.com)**, and **Natural Earth (https://www.naturalearth.com)**. Stay tuned for more!

If you would like to contribute, file a new issue in the **whosonfirst-data (https://github.com/whosonfirst-data/whosonfirst-data/issues/new)** repository or **drop us a line (mailto:hello@mapzen.com)**!

Photo Credit: ***sagesolar. (Flickr, CC BY 2.0) (https://flic.kr/p/SacxZh)***

· 19 September 2017 ·

Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

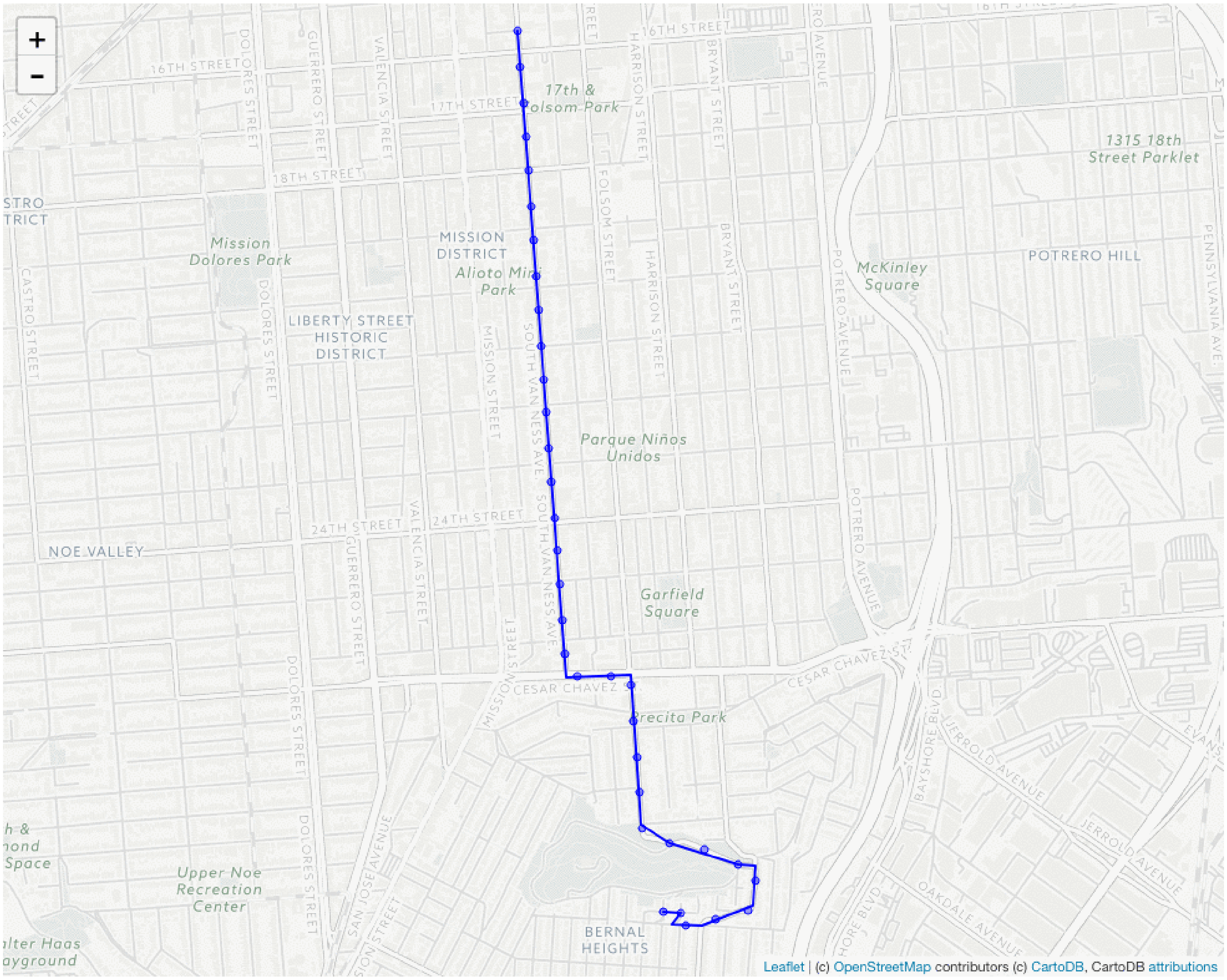
© 2017 Mapzen

A "data-driven" approach to improving map-matching, Part III -- Post-processing for Quality

routing (/tag/routing) mobility (/tag/mobility)

Since we announced ***our collaboration with the World Bank and more partners to create the Open Traffic platform (https://mapzen.com/blog/announcing-open-traffic/)****, we've been busy. We've shared* ***two (https://mapzen.com/blog/open-traffic-osmlr-technical-preview/) technical (https://mapzen.com/blog/osmlr-2nd-technical-preview/)*** *previews of the OSMLR linear referencing system. Now we're ready to share more about how we're using* ***Mapzen Map Matching (https://mapzen.com/blog/map-matching/)*** *to "snap" GPS-derived locations to OSMLR segments, and how we're using a data-driven approach to evaluate and improve the algorithms.*

So far, we've seen how Mapzen uses synthetic GPS data to **validate (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/23b90405f2d094c015a3438426a9a5d338b86c46/notebooks/map_matching_part_I.ipynb)** the our map-matching results, and also how we use those validation metrics to **optimize (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_II.ipynb)** our algorithm performance for each one of our Open Traffic partners. For the third installment of our series on data-driven map-matching, we'll discuss the post-processing we do to ensure that only the best-quality matches make their way into the Open Traffic datastore. Continue reading below, or follow along in the accompanying Jupyter Notebook **here (http://bit.ly/2ydWrvL)**.


+
-
16TH STREET
17TH STREET
18TH STREET
24TH STREET
CESAR CHAVEZ
17th &
Folsom Park
1315 18th
Street Parklet
Mission
Dolores Park
MISSION
DISTRICT
Alioto Mini
Park
LIBERTY STREET
HISTORIC
DISTRICT
McKinley
Square
POTRERO HILL
Parque Niños
Unidos
NOE VALLEY
Garfield
Square
Precita Park
Upper Noe
Recreation
Center
BERNAL
HEIGHTS
Leaflet | (c) OpenStreetMap contributors (c) CartoDB, CartoDB attributions

Background

As previously discussed **here (https://mapzen.com/blog/data-driven-map-matching/)**, Mapzen's map-matching algorithm relies on a kind of graphical model known as a Hidden Markov Model (HMM). Because HMM's are probabilistic in nature, the matches returned by the algorithm will always only be a best guess. Unfortunately, this best guess may still be quite far from the "truth" – AKA **wrong**.

To make sure we aren't serving up traffic conditions based on junk data, we would like some method for filtering out match results that just aren't realistic. One such approach we at Mapzen have developed is based on driving **driving speeds**.

Hypothesis

Although GPS can often have very poor positional accuracy, its temporal measurements are rarely off by more than a **nanosecond (http://www.atomic-clock.galleon.eu.com/support/gps-time-accuracy.html)**. Because of this temporal precision, we can assert that any candidate route matched between two GPS data points must traverse its path in exactly the amount of time elapsed between the measurements. What this means is that a truly wacky match – one that takes detours down side streets or doubles back on itself – is going to have a lot of additional ground to cover in the same amount of time. In other words, the driver on this hypothetical, wacky route would have to be going *very* fast.

The match error diagram from **Newson and Krumm (2009) (http://research.microsoft.com/en-us/um/people/jckrumm/Publications%202009/map%20matching%20ACM%20GIS%20camera%20ready.pdf)** helps illustrate how we might construct a speed-based error metric:

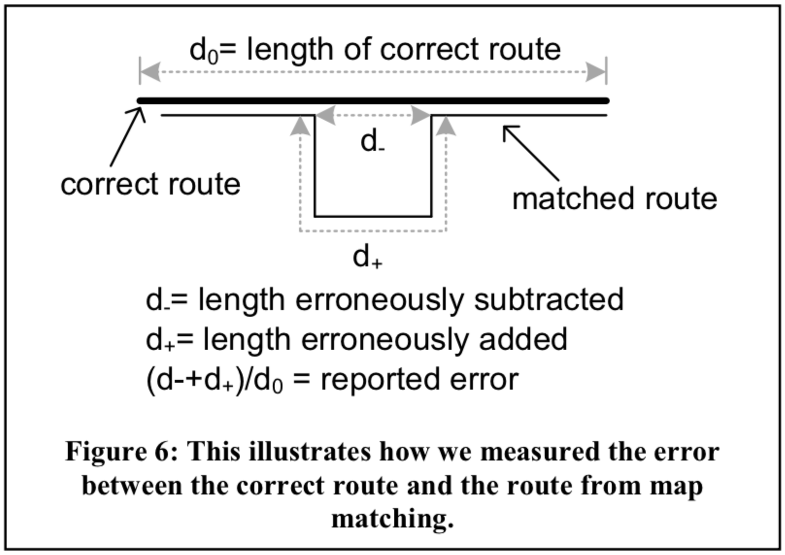

d_0= length of correct route
correct route
d_-
matched route
d_+
d_-= length erroneously subtracted
d_+= length erroneously added
$(d_- + d_+)/d_0$ = reported error

Figure 6: This illustrates how we measured the error between the correct route and the route from map matching.

From Newson and Krumm (2009)

Imagine that we have two GPS measurements on either end of the route shown above. Clearly, someone driving along the matched route would need to drive nearly twice as fast as someone on the correct route in order to traverse their paths in the same amount of time. Therefore, if we know that the average speed limit along the matched route is 50 km/h, and the speed we derive from our GPS measurements is 100 km/h, we probably have a bad match. Our hypothesis is that because bad matches are more likely to generate unreasonably high speeds relative to the known **speed limits (http://wiki.openstreetmap.org/wiki/Speed_limits)** along a route, **we should therefore be able to identify bad matches based on the magnitude of this speed error**.

Below, we attempt to learn from our simulated GPS traces what the optimal threshold for such a speed-based error metric might be.

Generating more routes and route scores!

For this analysis, we'll revert back to our original study area in San Francisco, CA:

```
mapzenKey = os.environ.get('MAPZEN_API')
cityName = 'San Francisco'
minRouteLen = 0.5     # specified in km
maxRouteLen = 1     # specified in km
numRoutes = 100
routes = val.get_routes_by_length(cityName, minRouteLen, maxRouteLen, numRoutes, mapzenKe
```

The previous bits of code should look familiar to you if you've been following along with our map-matching tutorials from the beginning (if not, check out **this (http://bit.ly/2h7nimh)** section of the first tutorial). The only difference here is that rather than scoring matches, we're interested in scoring speeds. Thus, we are only interested in the second object returned by the `get_route_metrics()` function, which correspond to the speed metric dataframe.

```
sampleRates = [1, 5, 10, 20, 30]    # specified in seconds
noiseLevels = np.linspace(0, 100, 21)    # specified in meters
_, speedDf, _ = val.get_route_metrics(cityName, routes, sampleRates, noiseLevels, saveRes
```

A random 5 entries from the speed metric dataframe:

```
speedDf.sample(5)
```

	matched	noise	pct_error	route_name	sample_rate	segment_id
206682	True	45.0	0.073008	Russell Ng_to_Lee Hecht Harrison	5.0	85569690162
271234	False	70.0	0.460363	Nan D Schlosberg_to_Total Logistics Resource Inc	10.0	825545579058

	matched	noise	pct_error	route_name	sample_rate	segment_id
174527	True	10.0	-0.030170	Sweet Sue's_to_All Metals Welding	1.0	122272716425
248699	True	55.0	-0.086480	Something Very Special_to_ECOR Clothing Enterp...	10.0	500531484576
46216	False	85.0	0.373544	AAABE Co_to_Street P	30.0	287662511753

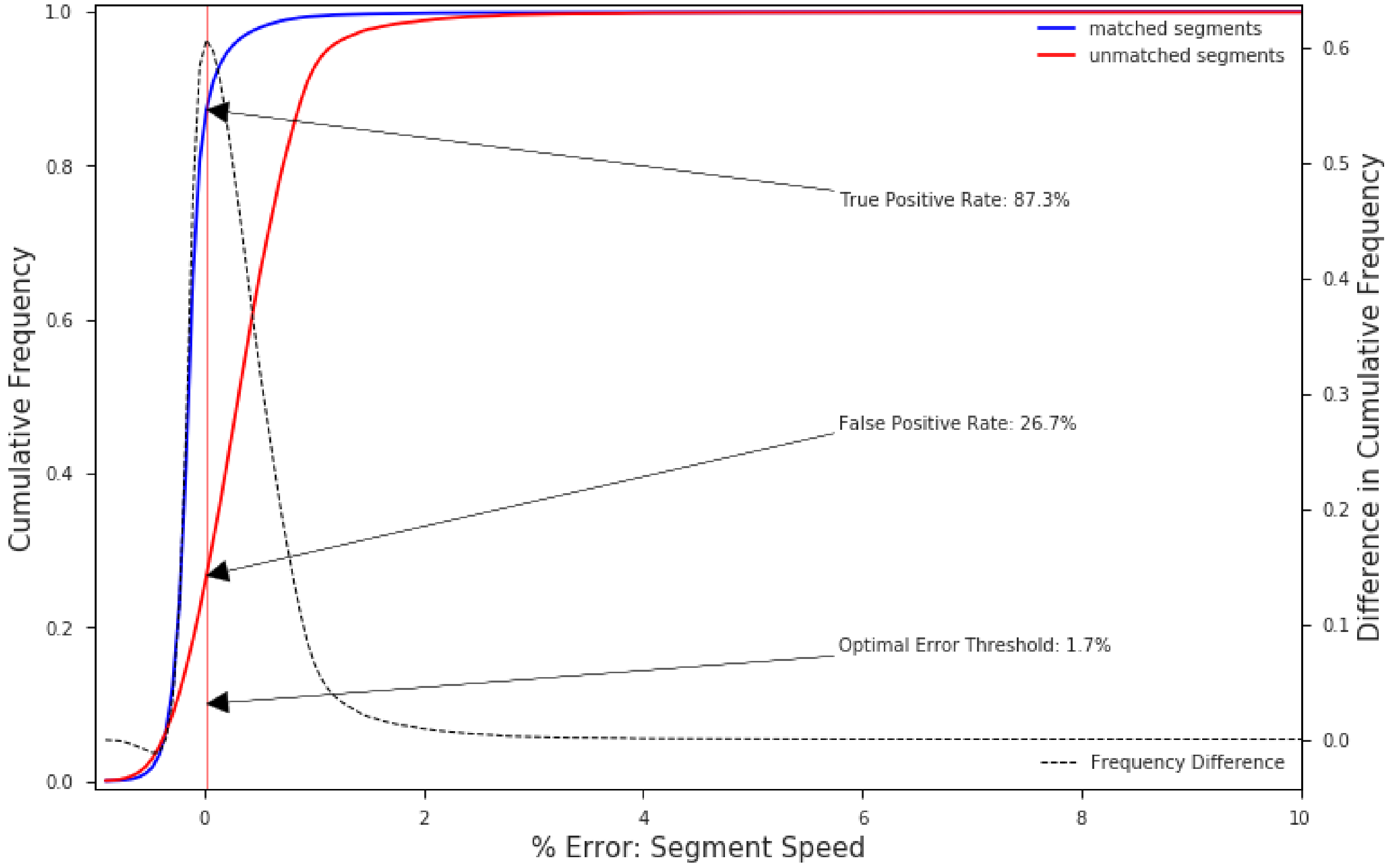
Calculate and plot the optimal speed error threshold

We define speed error in the following way:

$$\text{speed error} = \frac{\text{GPS-derived speed} - \text{known speed}}{\text{known speed}}$$

The optimal speed error threshold is defined by the error that maximizes the rate of true positives while minimizing the rate of false positives. We can graphically identify the optimal value as the point at which the cumulative distribution functions (**CDFs (https://en.wikipedia.org/wiki/Cumulative_distribution_function)**) of the error for matched and unmatched segments are farthest apart.

```
speedErrorThreshold = val.get_optimal_speed_error_threshold(speedDf, plot=True)
```



The chart above clearly shows that nearly 90% of correctly matched segments have a speed error rate of less than 1.7%, while nearly 75% of the unmatched segments, or false positive matches, lie above this value.

Food for thought: Weighted Penalty Functions

We need not default to the optimal speed error threshold, which treats false positives and false negatives as equally “bad”. For example, if a given customer/data provider cares more about mistakenly including bad data (Type I error) than missing good data (Type II error) a more conservative threshold can be chosen. The CDF plots above allow an analyst to visually inspect the impact that such discretionary adjustments will have on the outcome. These types of adjustments can also be performed in a programmatic fashion by assigning differential weights to our two types of error and thus shifting the optimal value itself.

Map-matching Accuracy with a Global Threshold

So, what kind of results can we expect from our “optimal” threshold? We’ll treat our threshold as a **binary classification** problem to compute the accuracy over some test data.

Generate test data

```
testRoutes = val.get_routes_by_length(cityName, minRouteLen, maxRouteLen, numRoutes, mapz
_, speedDf, _ = val.get_route_metrics(cityName, testRoutes, sampleRates, noiseLevels, sav
```

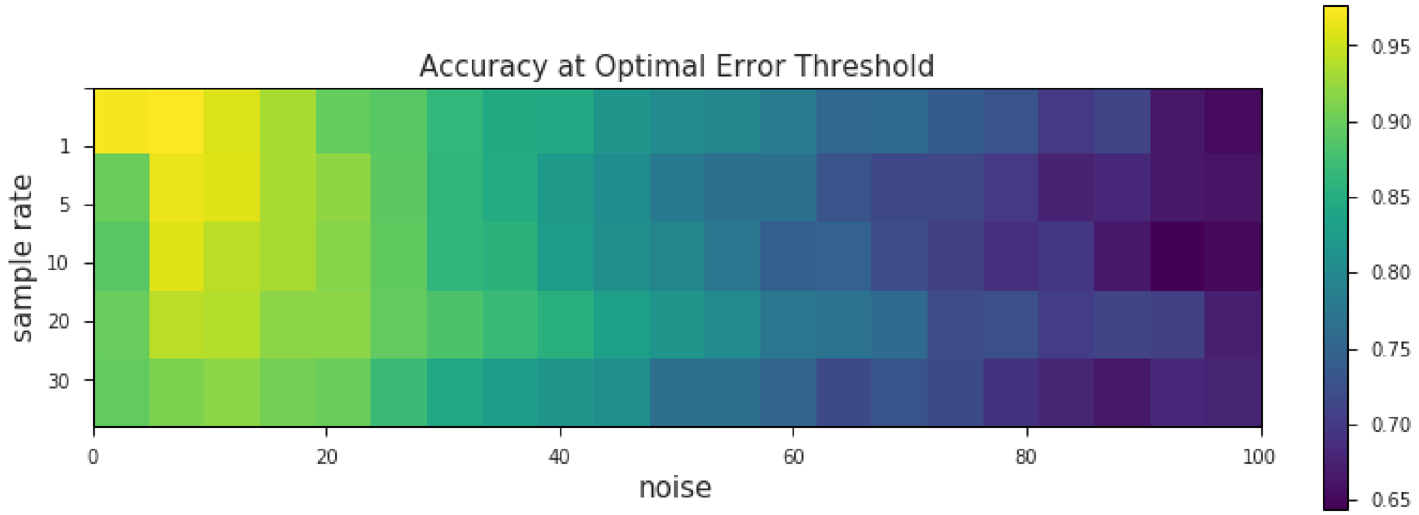
Score the results

Statistical accuracy measures the fraction of samples correctly identified as either true positives or true negatives, or in our case, matches and misses, respectively:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Population}} = \frac{\#\text{ Matches Kept} + \#\text{ Misses Discarded}}{\#\text{ Segments}}$$

Here's how the accuracies compare across different sample rates and noise levels of our test data:

```
optimalAcc = val.plot_accuracy_heatmap(speedDf, [speedErrorThreshold], sampleRates, noise
```



Using a single threshold across all levels of data quality yields some interesting results. Unsurprisingly we get better match accuracy at lower levels of noise (ignoring some of the weirdness that occurs at 0 m noise, where we haven't perturbed our data at all). We're well within the ~90% accuracy range where it counts, at 40 m of noise and below. What's more interesting here, however, are the striation-like patterns that emerge across the 5 sample rate levels. This is probably the same phenomemon we saw **before**

(http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_I.ipynb#3.-Visualize-the-Scores), where slower sample frequencies were shown to be more robust against higher levels of noise.

Map-matching Accuracy with Rate-based Thresholds

As we saw in the case study of **A-OK Rides (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_II.ipynb#4.-Case-study:-%22A-OK-Rides%22)**, a fixed GPS sample rate can easily be set by a given data provider. If we know what this sample rate will be ahead of time, we can tune the speed-error threshold accordingly, and avoid using a globally-defined threshold for all sample rates. This should hopefully improve the overall accuracy of our matches.

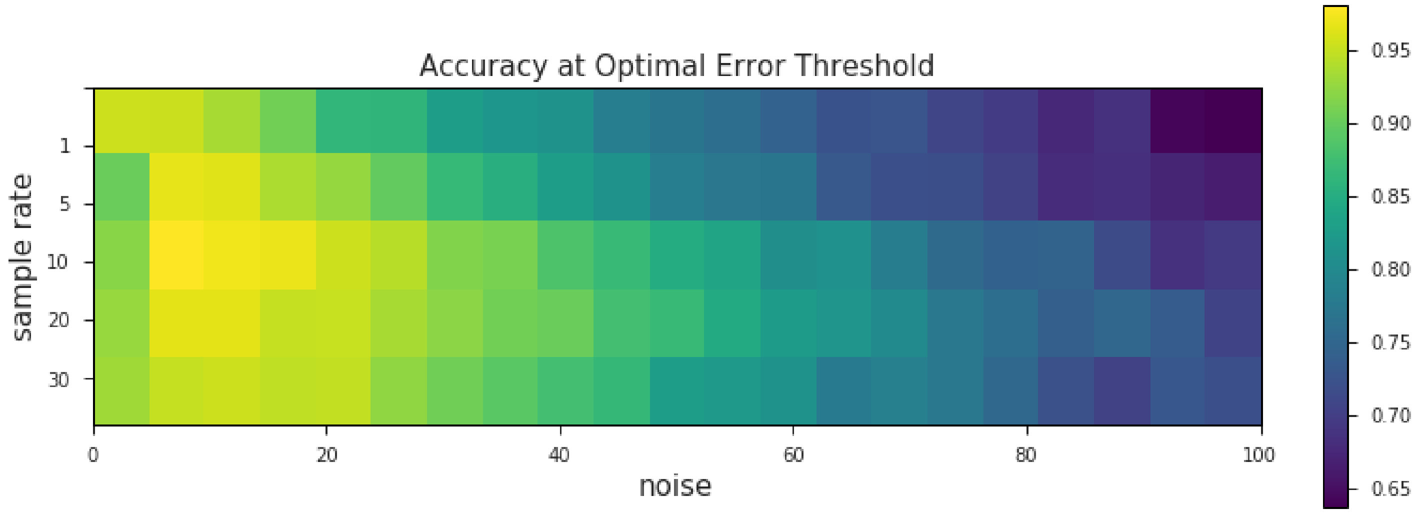
```
thresholdRateDict = {}
thresholdsByRate = []
for r, rate in enumerate(sampleRates):
    df = speedDf[speedDf['sample_rate'] == rate]
    threshold = val.get_optimal_speed_error_threshold(df, plot=False, saveFig=False)
    thresholdsByRate.append(threshold)
    thresholdRateDict[str(int(sampleRates[r]))] = threshold
```

We now have 5 different thresholds – one for each sample rate – instead of one:

```
{'1': 0,
 '10': 0.066631693373000012,
 '20': 0.053864199086380071,
 '30': 0.074422049226850029,
 '5': 0.020375927157593399}
```

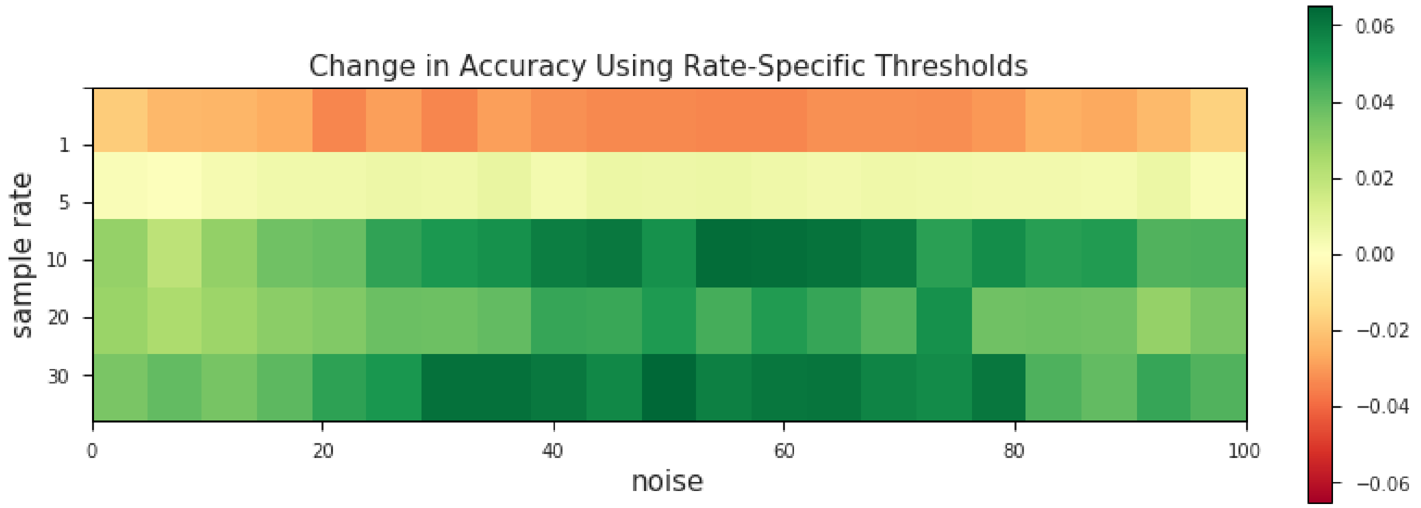
We can now recalculate the statistical accuracy using the rate-specific thresholds…

```
rateSpecificAcc = val.plot_accuracy_heatmap(speedDf, thresholdsByRate, sampleRates, noise
```



…and visualize the change between the two:

```
val.plot_change_in_acc(oneSizeFitsAll, accBySampleRate, sampleRates, noiseLevels)
```



These charts suggest that **prior knowledge of GPS sample rate can result in accuracy gains of up to ~6% when used in conjunction with the optimal speed error threshold**.

*** N.B. - there are a number of reasons why the accuracy of 1 second sample rate matches might decrease when filtering based on the rate-specific threshold. The most likely explanation is that at a 1 s sample rate there either aren't enough false positives to pick a reliable threshold or the GPS-derived speeds of the false positives just aren't different enough from those generated by true positives.*

Visually Inspect Speed-Based Filtering

Let’s see our new junk-data filtering mechanism in action! Well test it for a hypothetical provider who collects GPS data at a 10 s sample rate with a 60 m standard deviation of noise:

Get more routes

```
vizRoutes = val.get_routes_by_length(cityName, 0.5, 1, 50, mapzenKey)
```

Pass the speed error threshold to `get_route_metrics()` as a argument!

```
sampleRates = [1, 5, 10, 20, 30]    # specified in seconds
noiseLevels = np.linspace(0, 100, 21)    # specified in meters
noiseLevels = [60]
sampleRates = [10]
```

```
matchDf, speedDf, densityDf = val.get_route_metrics(cityName,
    vizRoutes, sampleRates, noiseLevels,
    speedErrThreshold=thresholdRateDict['10'])
```

Visualize the thresholding

In the map below we’ve styled the output of our sample routes in order to highlight the segments and their speeds that will either make their way into the datastore or get thrown out. This is simply a way of visualizing a process that in practice would occur entirely behind-the-scenes, and in the following order:

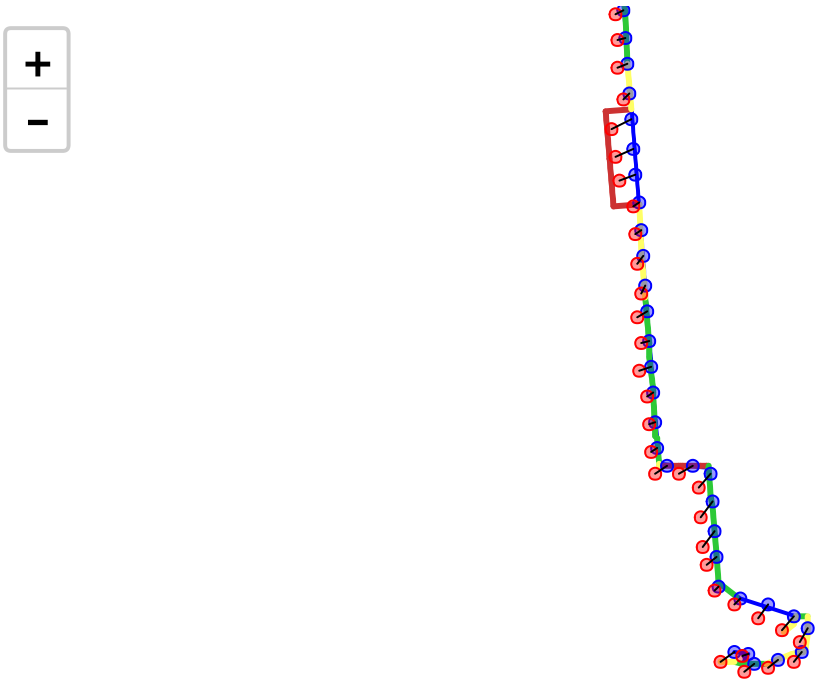
1. Generate a random route (▬) to serve as our “ground truth”
2. Resample points (◉) along the route at a user-defined sample rate
3. Add a semi-random amount of noise to **simulate GPS (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_I.ipynb#2.-Iterate-Through-Routes,-Generate-Fake-GPS,-and-Score-the-Matches)** (◉)
4. Apply the predetermined speed-error threshold:
    - Send segments that do not exceed the threshold (▬) to the Open Traffic datastore
    - Filter out segments that exceed the threshold (▬)
    - Filter out segments for whom speeds could not be calculated (▬)

Here we'll get the list of all the geojson files that were created when we synthesized new GPS traces…

```
dataDir = '../data/'
geojsonList = [trace for trace in os
               .listdir(dataDir) if trace.endswith('json')]
```

…and run the following cell repeatedly to iterate through the routes to view them, along with their matches, on a slippy map!

```
fname = dataDir + geojsonList.pop()
m = val.generate_route_map_2(fname, 17)
m
```



Leaflet (http://leafletjs.com) | (c) OpenStreetMap (http://www.openstreetmap.org/copyright) contributors (c) CartoDB (http://cartodb.com/attributions), CartoDB attributions (http://cartodb.com/attributions)

There's a lot to look at here, but there are two important takeaways illustrated by the map above:

1. The section of the matched route that deviates from the “ground truth” has been **correctly identified** by our speed-error threshold as a **bad match** as indicated by the red color. In other words, the threshold seems to be doing its job.
2. Zooming in on the map, it seems our threshold has *misclassified* the east-west portion of the route along Cesar Chavez St. as a bad match, when it clearly is not. This suggests that our threshold may be doing its job a bit **too well**, at least in this instance.

*** N.B. - The section of the route on the northern face of Bernal Hill doesn’t show a colored match due to an idiosyncracy in how OSMLR segments (map-matching results) get mapped to Valhalla traces. Although it effects our ability to visualize the process, it doesn’t impact the results in practice.*

Results

Ultimately, the speed-error threshold seems to be doing its job of identifying the truly egregious, detour-style down-and-back mismatches that are caused by noisy GPS data and very difficult to avoid with map-matching alone. Although the “optimal” threshold we derived may be on conservative side (i.e. lends itself to Type II error), it’s not hard to image why certain Open Traffic partners may wish to err on the side of caution. We, personally, would much rather have gaps in coverage than get our users stuck in traffic. As far as what our data providers/partners/users prefer, well, we think thats a conversation worth having. What we’ve shown here is a methodology for approaching that conversation in a more **informed** and **statistically rigorous** way.

· 20 September 2017 ·

Max Gardner

Max is a summer 2017 intern on the Mapzen Mobility team. A PhD student at UC Berkeley, he will continue to support the Cavs even if LeBron leaves Cleveland again because, after all, Max left too.

© 2017 Mapzen

A "data-driven" approach to improving map-matching: Part IV — Map-Matching and the Built Environment

routing (/tag/routing) mobility (/tag/mobility)

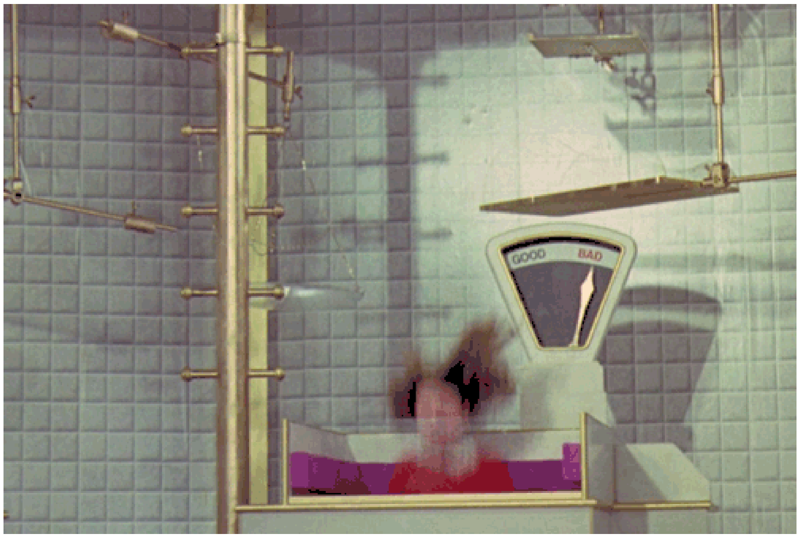
Since we announced our collaboration with the World Bank and more partners to create the Open Traffic platform (https://mapzen.com/blog/announcing-open-traffic/), we've been busy. We've shared two (https://mapzen.com/blog/open-traffic-osmlr-technical-preview/) technical (https://mapzen.com/blog/osmlr-2nd-technical-preview/) previews of the OSMLR linear referencing system. In this series of blog posts (/tag/data-driven), we've been sharing more about how we're using Mapzen Map Matching (https://mapzen.com/blog/map-matching/) to "snap" GPS-derived locations to OSMLR segments, and how we're using a data-driven approach to evaluate and improve the algorithms.

Background

So far we have seen that map-matching accuracy can be affected by both variables we can fix over time (e.g. GPS sample rate) and those we cannot (e.g. GPS accuracy/noise). The ability to predetermine a *sample rate* allows us measure its impact on match error and **make recommendations (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_I.ipynb#4.-Results)** to our data partners regarding the fidelity of their data. Conversely, because GPS *positional accuracy* is so highly variable, we must make a more **nuanced decision (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_II.ipynb#b)-Customizing-our-search-space)** about the range of expected noise levels for which we wish to optimize the algorithm.

A third class of variable that impacts map-matching performance consists of those related to the **built environment**. Like GPS sample rate, built environment characteristics remain fixed in the short term, but like GPS accuracy, we have no control over them. Thus, despite our intuition about the impact of something like road network density on match performance, we cannot recommend that a data provider build a less dense road network. Nevertheless, it is still worthwhile to quantify the relationship between built environment characteristics and map-matching performance for two principal reasons:

1. to help **set expectations** for both our users and data providers based on characteristics specific to their region (i.e. determine the upper limit of performance)
2. to make more **informed decisions** about how conservative we want to be when **separating the good data from the bad (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_III.ipynb?flush_cache=true#2.-Calculate-and-plot-the-optimal-speed-error-threshold)**:



***Good vs bad (https://giphy.com/gifs/queue-willy-wonka-and-the-chocolate-factory-oVIJX9HoKYI8w)** data.*

In the analysis that follows, we will rely on **road network density** as a proxy for the character of the built environment. Road network density reflects a number of highly relevant environmental characteristics including land use types, city block size, and the prevalence of **urban canyons (https://en.wikipedia.org/wiki/Street_canyon)**.

The analysis will proceed in two parts. First, we will **assess the match rates** of individual road segments **as a function of the local road network density** at that segment. Second, having established the relationship between density and map-matching performance on the smallest of scales, we will undertake a case study **comparing the regional performance across two cities** with very different road network structures. Continue reading below or following along in the accompanying Jupyter Notebook **here (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_IV.ipynb)**.

Segment-level Analysis

Mapzen's Valhalla routing engine measures road network density as as function of the total length of the road network contained within a 2 km circle around a given node on the network. In this way we derive measures of road network density at each individual segment of a network. The densities are normalized across a range of integers from 0 to 15.

To compute density measures, we can use the Mapzen Map Matching service. For more information, see the `trace_routes` action and the `density` attribute in the **API documentation (https://mapzen.com/documentation/mobility/map-matching/api-reference/)**.

In this analysis, we will sample routes across a variety of different U.S. cities we ensure that our segments will span the entire range of possible densities. Here we have selected 15 cities of varying sizes and geographies:

```
cities = ['New York City', 'Los Angeles', 'Dallas, TX', 'Miami, FL', 'Levittown, NY', 'Pr
         'Flint, MI', 'Chicago, IL', 'Boulder, CO', 'Lincoln, NE', 'Charleston, SC', 'Por
         'Philadelphia, PA', 'Baton Rouge, LA', 'Knoxville, TN']
```

Generate routes

```
import validator.validator as val
minRouteLen = 1     # specified in km
maxRouteLen = 5     # specified in km
numRoutes = 50
allRoutes = []
for cityName in cities:
        routeList = val.get_routes_by_length(cityName, minRouteLen, maxRouteLen, numRoute
        allRoutes += routeList
```

Generate scores for all routes

In this disaggregate analysis, we don't care about what city or route each segment match came from. Instead, we'll pool all of our segments to determine whether those that were correctly matched have a different distribution of road network densities than those segments that were not.

```
sampleRates = [1, 5, 10, 20, 30]    # specified in seconds
noiseLevels = [0, 20, 40, 60, 80, 100]   # specified in meters
```

For now, we're only interested in density of each individual segment and whether or not it was matched correctly. As such, we're only interested in the third object in the `get_route_metrics()` function:

```
_, _, densityDf = val.get_route_metrics('allCities', allRoutes, sampleRates, noiseLevels,
```

The contents of the density dataframe are summarized below:

```
# random 5 rows of dataframe
densityDf.sample(5)
```

	density	sample rate	noise	matched
0	13	1	0	True
1	13	1	0	True
2	13	1	0	True
3	13	1	0	True
4	13	1	0	True

```
# summary of dataframe
densityDf.describe(include='all')
```

	density	sample rate	noise	matched
count	866970.000000	866970.000000	866970.000000	866970
unique	NaN	NaN	NaN	2
top	NaN	NaN	NaN	True

	density	sample rate	noise	matched
freq	NaN	NaN	NaN	790219
mean	10.166303	13.200000	50.000000	NaN
std	2.940786	10.533761	34.156522	NaN
min	2.000000	1.000000	0.000000	NaN
25%	8.000000	5.000000	20.000000	NaN
50%	10.000000	10.000000	50.000000	NaN
75%	12.000000	20.000000	80.000000	NaN
max	15.000000	30.000000	100.000000	NaN

Compare the distribution of road network densities by match type

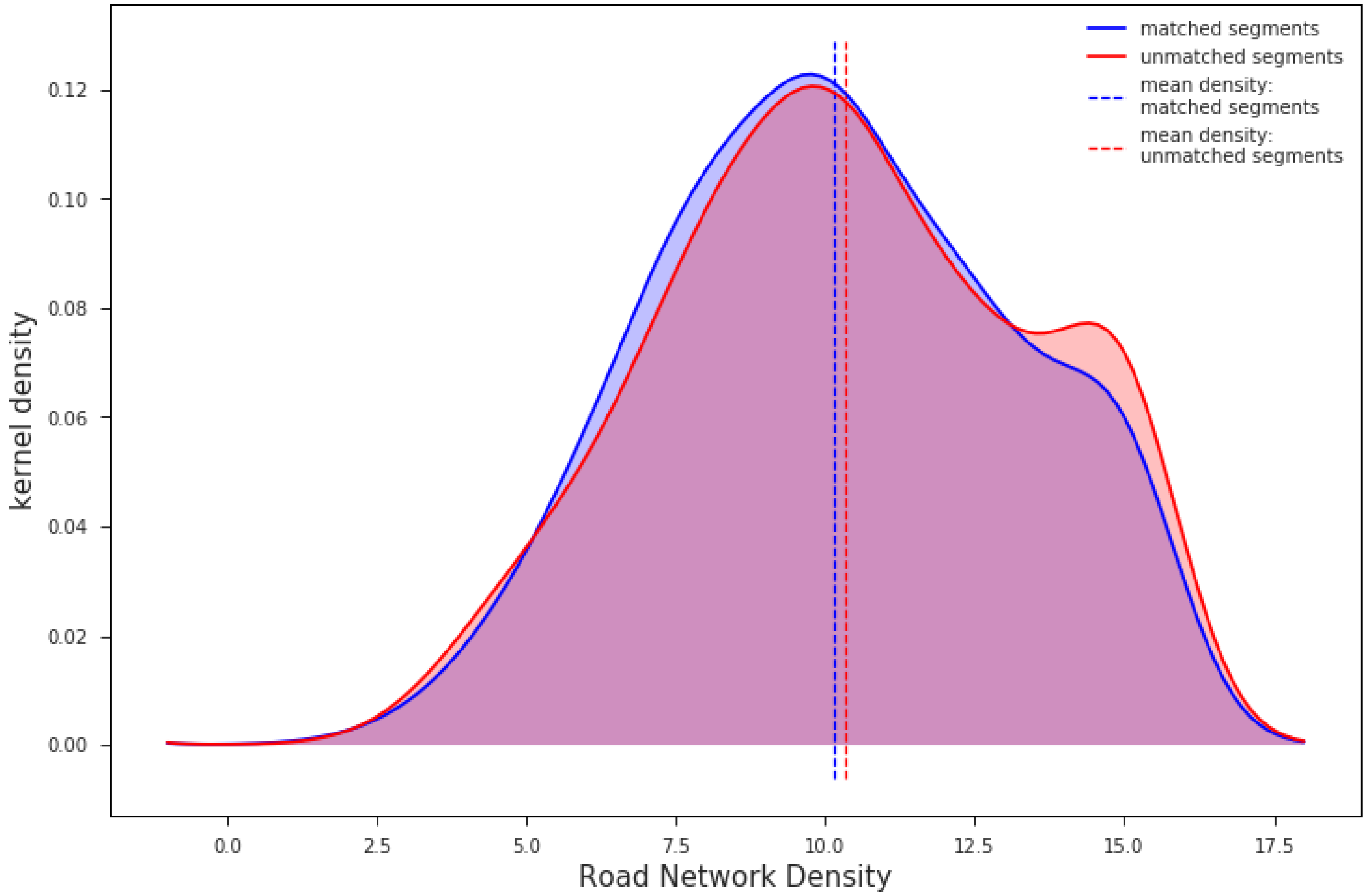
Kernel density** estimates (KDEs) provide a nice way to estimate and visualize the distribution of a “random” variable. A KDE is estimated, in essence, by applying a rolling average to a frequency distribution or histogram, with the “kernel” loosely defined as the window used to compute that rolling average.

Our hypothesis, derived from the intution that higher densities are more prone to erroneous matches, is that the KDE of road network densities around successfully matched segments should be skewed towards lower values relative to the KDE of densities derived from unmatched segments.

*** N.B. - the word density here refers to a probability density, not to be confused with road network density which is the variable we are modeling*

The KDE’s for our two distributions are shown here:

```
val.plot_density_kdes(densityDf)
```

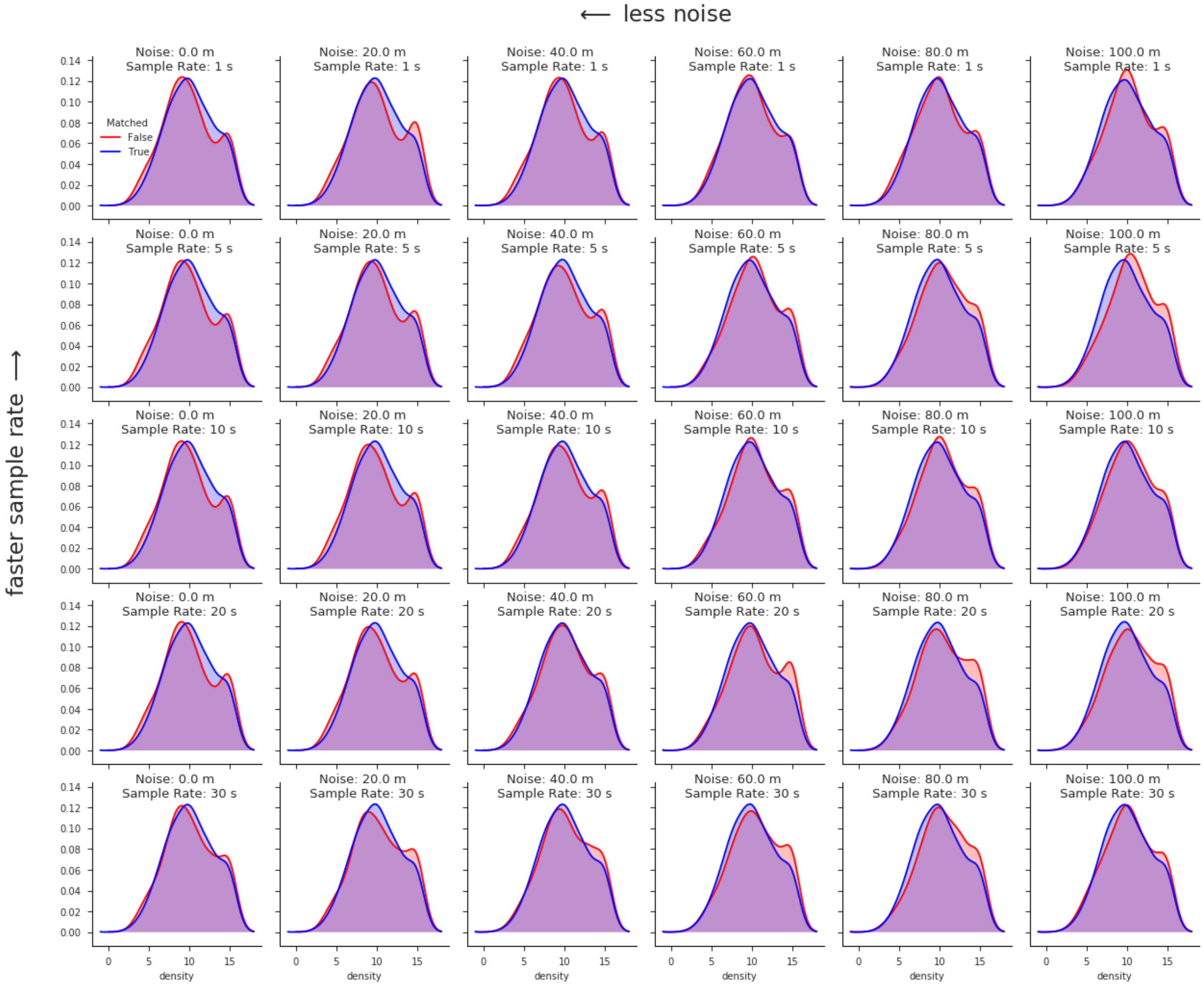


The chart above shows that the distribution of road network densities are actually much more similar than we might expect. It suggests that road network density on its own would not make a good predictor of match rate.

There is, however, a clear difference between the distributions, albeit somewhat small in magnitude. Most importantly, this difference is in the direction that matches our intuition: unmatched segments have a higher mean road network density than their matched counterparts. Given that the distributions were calculated over such a wide swath of sample data – nearly 900,000 individual road segments from 750 routes in 15 different cities – we can be confident that the trend is real.

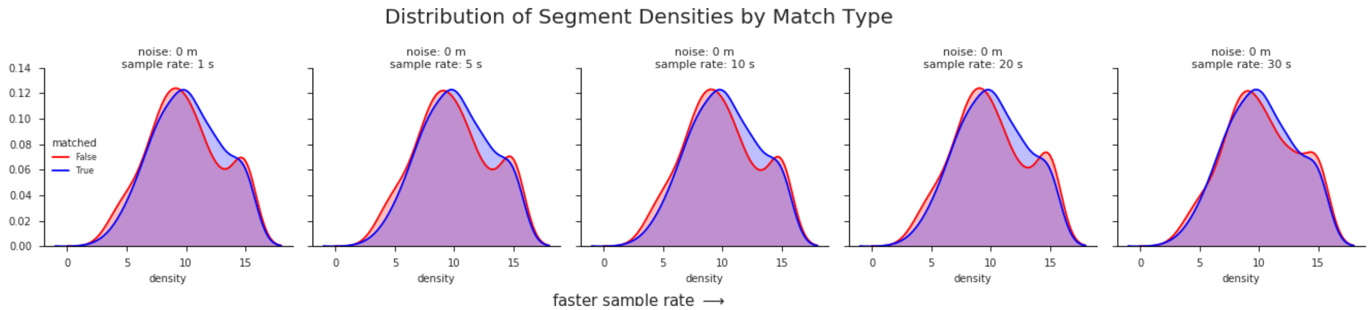
Drilling down a bit deeper, we see that this trend persists across almost all different GPS sample rates and noise levels. It is most pronounced at higher noise levels and slower sample rates:

```
val.plot_density_kdes_gridded(densityDf, saveFig=True)
```



(https://s3.amazonaws.com/mapzen-assets/images/map-matching-built-env/output_32_0.png) *click image to zoom*

Another view of the KDE’s in action, iterating through different levels of noise:



Normalize the distributions

Although the evidence so far does suggest a **weakly negative** influence of road network density on map-matching performance, the actual empirical relationship between the two is somewhat obscured by the KDE's. Since KDE plots are essentially visualizing a frequency distribution, we may be missing important visual information in the tail ends of the distributions where there simply aren't as many data points.

Since we don't really care about the frequency with which different road network densities occur, but rather how these densities impact map-matching performance when they *do* occur, we can normalize our results by the number of segments we observed at each road network density level (1 - 16).

```
freqDf = val.normalize_densities(densityDf, sampleRates, noiseLevels)
```

The first 10 rows from our new dataframe:

```
freqDf.head(10)
```

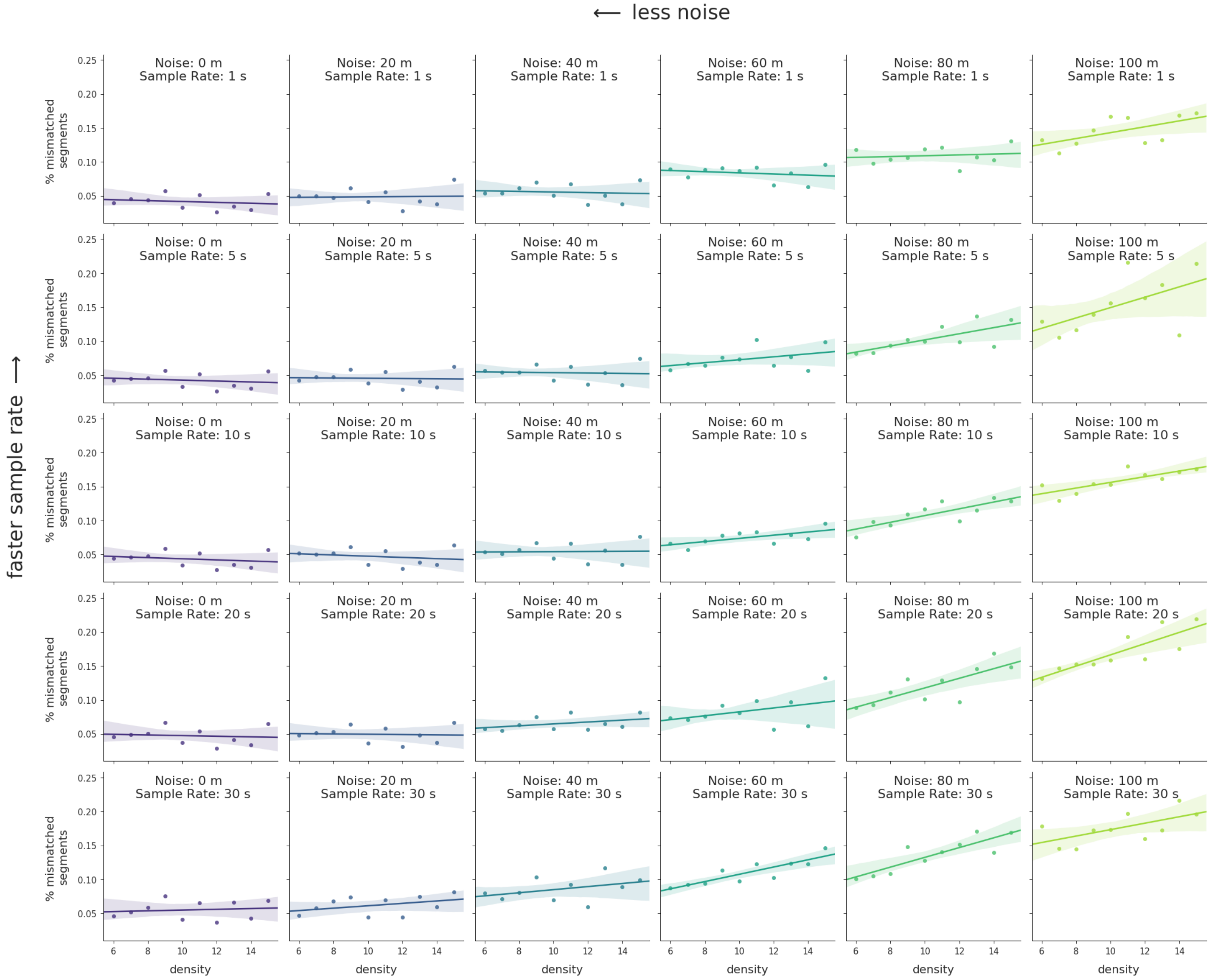
	sample rate	**noise**	**density**	**matched**	**frequency**
segment					
0	1	0	6	False	0.039440
1	1	0	6	True	0.960560
2	1	0	7	False	0.045342
3	1	0	7	True	0.954658
4	1	0	8	False	0.044005
5	1	0	8	True	0.955995
6	1	0	9	False	0.057152
7	1	0	9	True	0.942848
8	1	0	10	False	0.033078

	sample rate	noise	density	matched	frequency
segment					
9	1	0	10	True	0.966922

Now that we are dealing with the normalized match frequencies, there is no need to visualize both the matched and unmatched rates – they will always sum to 1. Instead, we will focus our attention on the percentage of **unmatched segments** at each discrete variable level, since this number more directly relates to the relationship between road network density and match error.

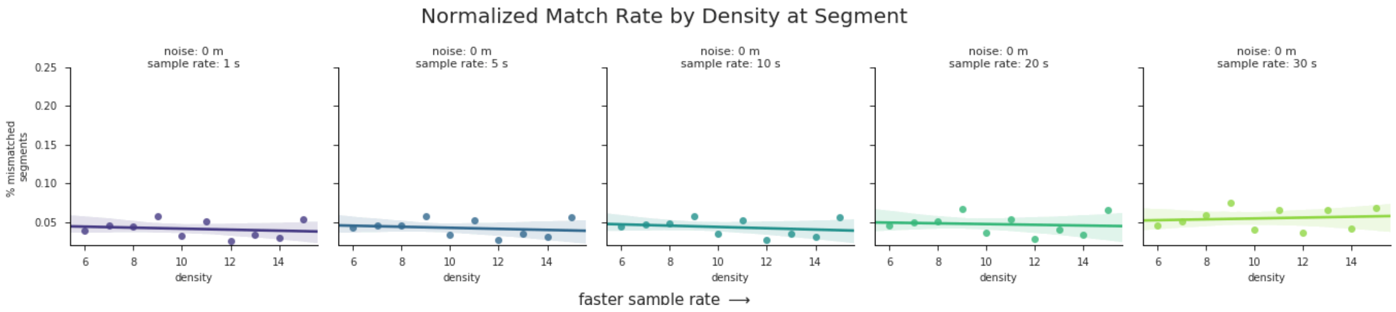
Plotting just the normalized frequencies of segment types where `matched == False` we arrive at the following:

```
val.plot_density_regressions(freqDf)
```



(https://s3.amazonaws.com/mapzen-assets/images/map-matching-built-env/output_39_0.png) *click image to zoom*

And the same charts animated, iterating through noise levels:



These charts highlight a number of trends that are worth pointing out:

1. There is a clear positive correlation between road network density on the `X` axes and error rate on the `Y` axes for the majority of noise levels and sample rates. In short, **the greater the density of the road network the worse the map-matching results will be**.
2. The strength of this relationship increases with increasing noise, from nearly 0 correlation at the lowest levels of noise to a trendline of nearly 3% error per unit of density at 100 m of GPS noise. In terms of real numbers, a route with an average density of 12 could expect up to **12% greater** mismatches than a route traversed in an area with a road network density of ~8.
3. Overall error rate increases with GPS noise. This is actually a trend we *aren't* interested in, since we've already assessed the impact of GPS noise on match accuracy **at length (http://nbviewer.jupyter.org/github/opentraffic/reporter-quality-testing-rig/blob/master/notebooks/map_matching_part_I.ipynb?flush_cache=true#3.-Visualize-the-Scores)**. Thus the relative positions of the regression lines along the `Y` axes in the charts only serve as a distraction. In the next section, we'll fix that problem.

Data standardization

The process of data *standardization* occurs in two steps:

1. **Center** a dataset about its own mean (i.e. μ = 0)
2. **Scale** the dataset relative to its own standard deviations (i.e. σ = 1)

By standardizing our data across noise levels, we'll eliminate in this way, we can visualize our data on a single set of axes that highlights just the change in slope between the trendlines at varying levels of GPS noise:

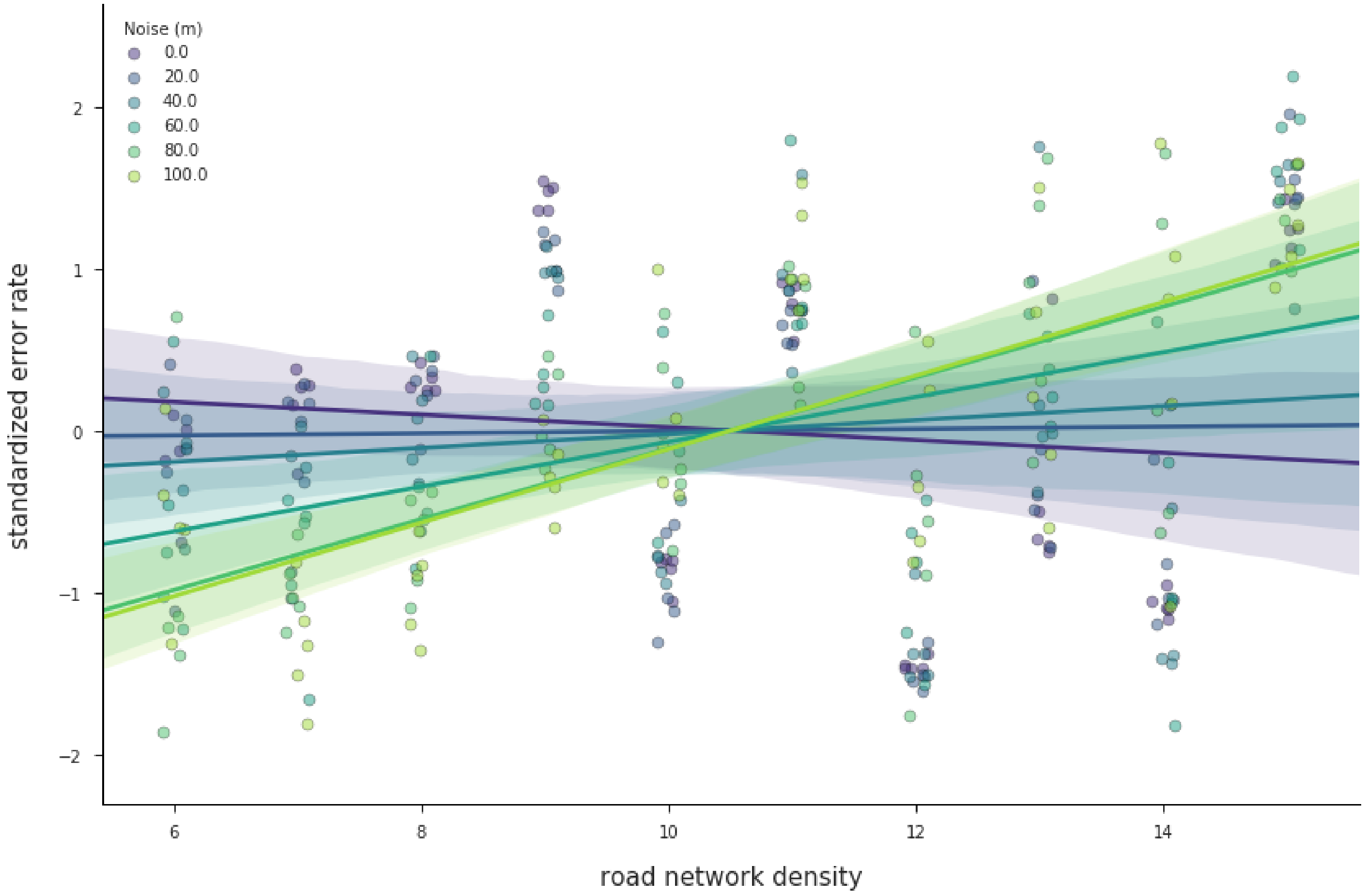
```
stdDf = val.standardize_densities(densityDf)
```

The first 5 rows of the standardized dataframe:

	sample rate	density	noise	mismatch rate	stddev mismatch rate	mean mismatch rate	normed mismatch
0	1	6	0.0	0.039440	0.010492	0.041373	-0.18
1	1	7	0.0	0.045342	0.010492	0.041373	0.38

	sample rate	density	noise	mismatch rate	stddev mismatch rate	mean mismatch rate	normed mismatch
2	1	8	0.0	0.044005	0.010492	0.041373	0.25
3	1	9	0.0	0.057152	0.010492	0.041373	1.50
4	1	10	0.0	0.033078	0.010492	0.041373	-0.79

```
val.plot_std_densities(stdDf)
```



With standardized data, we can more clearly see how road network density exerts an increasingly negative effect on segment match error rate as the level of GPS noise increases. The actual values on the `Y` axis, however, are unitless, and therefore less easily interpreted.

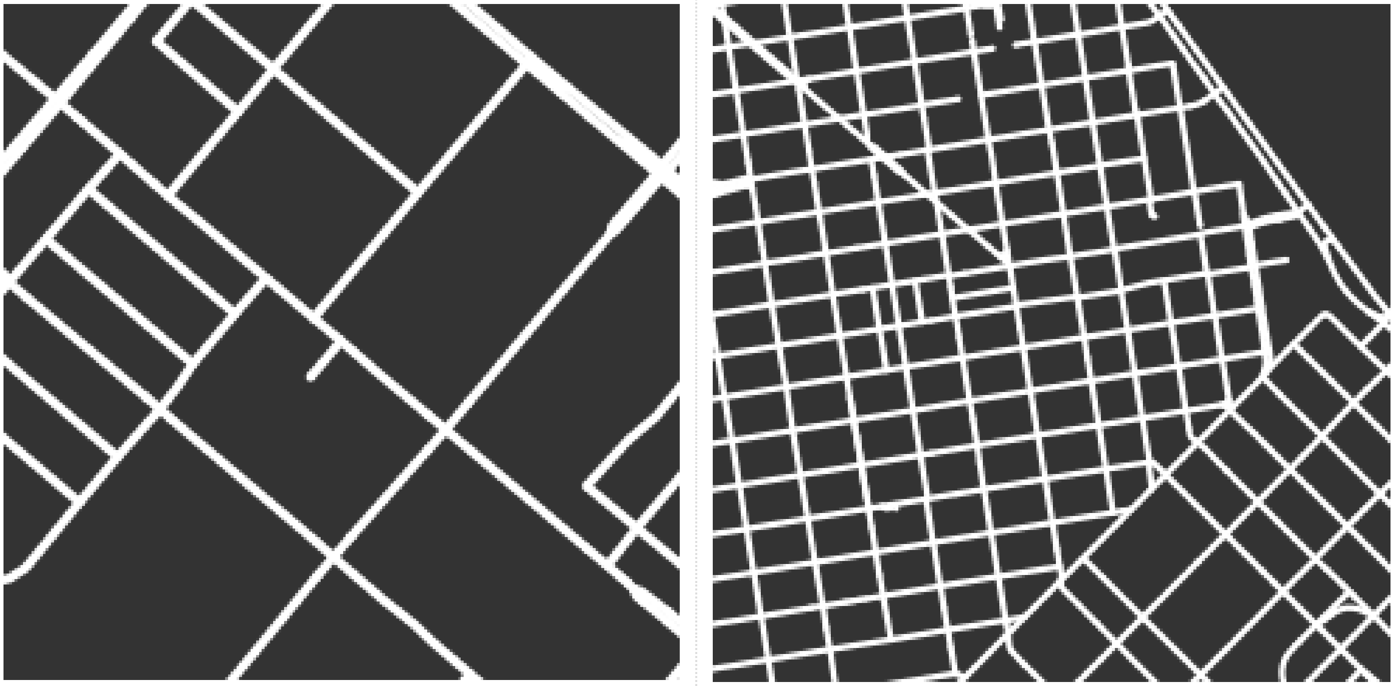
For some bonus eye candy, we can represent the same data with an interative parallel coordinates plot. This interactive mode of visualization allows us to dynamically filter our data to see how certain variables interact with others. In graph below, drag the magenta line up and down on the road network density axis and watch how the range of standardized error rates change in response:

Loading graph

You can also create new filters by clicking and dragging on the other axes to draw a filter of an arbitrary size. Give it a try!

Regional Analysis: Irvine, CA vs. San Francisco, CA

Clearly the built environment has a significant impact on the likelihood of matching a GPS measurement to its correct road segment. This is a particularly notable finding not just because an end user might expect worse performance in the denser portions of their route, but because two different data providers might expect entirely different performance based on the general character of their service territories. To illustrate this point, we will compare map-matching performance for two cities in California with very different geographies.



Irvine, CA - avg. density: ~8

San Francisco, CA - avg density: ~12

1 Sq. Mile from each road network (generated using **Geoff Boeing's (http://geoffboeing.com/)** excellent **osmnx (https://github.com/gboeing/osmnx)**). For an in-depth comparison of these two regions, check out the case study in the last section of this document.

Irvine, California

Irvine is a an affluent, planned city in southern California, just south of Los Angeles. It was planned by the **Irvine Company (https://en.wikipedia.org/wiki/Irvine_Company)**, which continues to control development in the area. Irvine is home to UC Irvine, which is also the city's largest employer. The city has a population density of 4,057.85/sq mi according to **Wikipedia (https://en.wikipedia.org/wiki/Irvine,_California)**.

```
cityName = 'Irvine, CA'
routeList = val.get_routes_by_length(cityName, 1, 5, 100, mapzenKey)
noiseLevels = np.linspace(0, 100, 21)
sampleRates = [1, 5, 10, 20, 30]
irvMatchDf, _, _ = val.get_route_metrics(cityName, routeList, sampleRates, noiseLevels, s
```

San Francisco, California

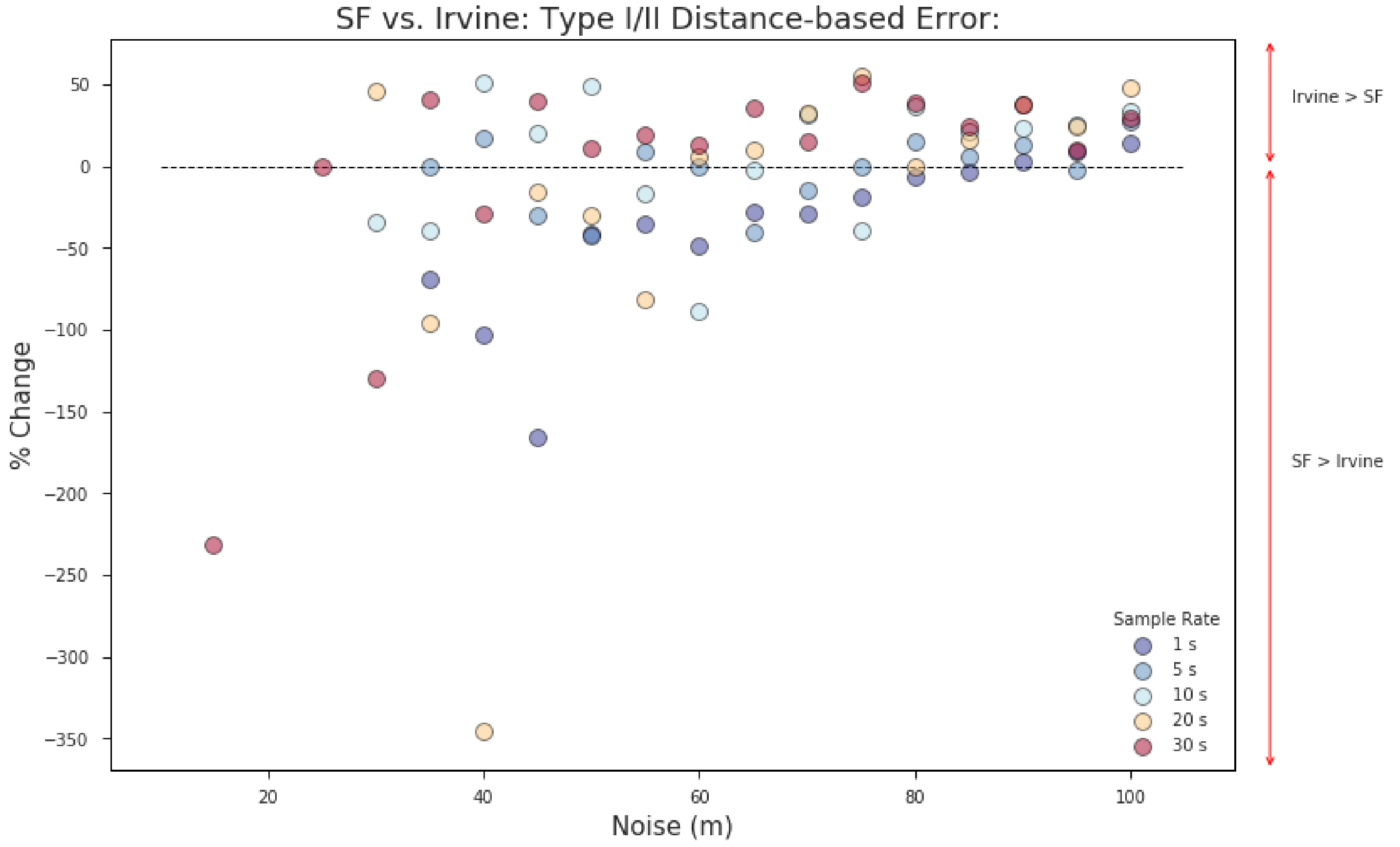
San Francisco is the second most densely populated major city in the United States, with a density of roughly 17,246.4 people per square mile according to **Wikipedia (https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population_density)**. It is entirely contained within a peninsula roughly 7 miles by 7 miles, and serves as the major west coast hub of American finance and technology.

```
cityName = 'San Francisco'
routeList = val.get_routes_by_length(cityName, 1, 5, 100, mapzenKey)
noiseLevels = np.linspace(0, 100, 21)
sampleRates = [1, 5, 10, 20, 30]
sfMatchDf, _, _ = val.get_route_metrics(cityName, routeList, sampleRates, noiseLevels, sa
```

Comparison

In the plot below we define "% change" as the difference between the Irvine and SF error rates divided by the Irvine error rate for each simulated sample rate and noise level:

```
val.plot_regional_comparison(sfMatchDf, irvMatchDf, sampleRates, cityNameA='SF', cityName
```



*** N.B. - Because the equation can't be evaluated when the denominator is zero, there aren't data points for many of the sample rates at the lowest levels of noise where the Irvine error rate == 0.*

A few takeaways from the chart above:

1. The dominant trend is that as GPS noise levels increase, map-matching performance **increases** for the less dense region (Irvine) relative to the denser region (San Francisco).
2. The difference in density seems to have the biggest impact on slower sample rates. With a 30 s sample frequency, the low-density region outperforms the higher density region with as little as 35 m of noise while the same trend doesn't occur at the 1 s sample rate until the GPS error is 90 m or more. This suggests that data providers and TNC's with particularly dense service territories can **make up for their structural disadvantages by using faster sample rates.**
3. San Francisco outperforms the less dense Irvine region at lower noise levels, especially for faster sample rates. This may be due to a number of factors (better OSM data, more one-way roads, better **parameter tuning (https://mapzen.com/blog/data-driven-map-matching/)**, etc.), but it suggests that there are other differences betweent the two regions apart from road network density that are likely effecting the results shown here.

Conclusions

In this analysis, we've tried to examine the extent to which certain built environment characteristics impose structural limitations on map-matching performance. Although it may seem obvious, it is extremely important that data providers and end users alike understand that **denser land use types will negatively impact vehicle match accuracies** for even the most sophisticated map-matching techniques. Just as important, however, is understanding what we can do about it. Increased GPS sampling rates seems to be one effective approach that data providers have at their disposal for combatting this effect, and surely others exist as well.

Thanks for reading all four posts (/tag/data-driven) in this series. To try map-matching yourself, open the Mapzen Mobility Explorer (https://mapzen.com/mobility/explorer) or read Mapzen Map Matching API documentation (https://mapzen.com/documentation/mobility/map-matching/api-reference/). To dive deeper into the code behind these map-matching analyses, see this repository on GitHub (https://github.com/opentraffic/reporter-quality-testing-rig/). And to learn more about Open Traffic and Mapzen's continued work with GPS probe data, please watch this blog or sign up for our newsletter (https://mapzen.com/newsletter-signup/).

· 29 September 2017 ·

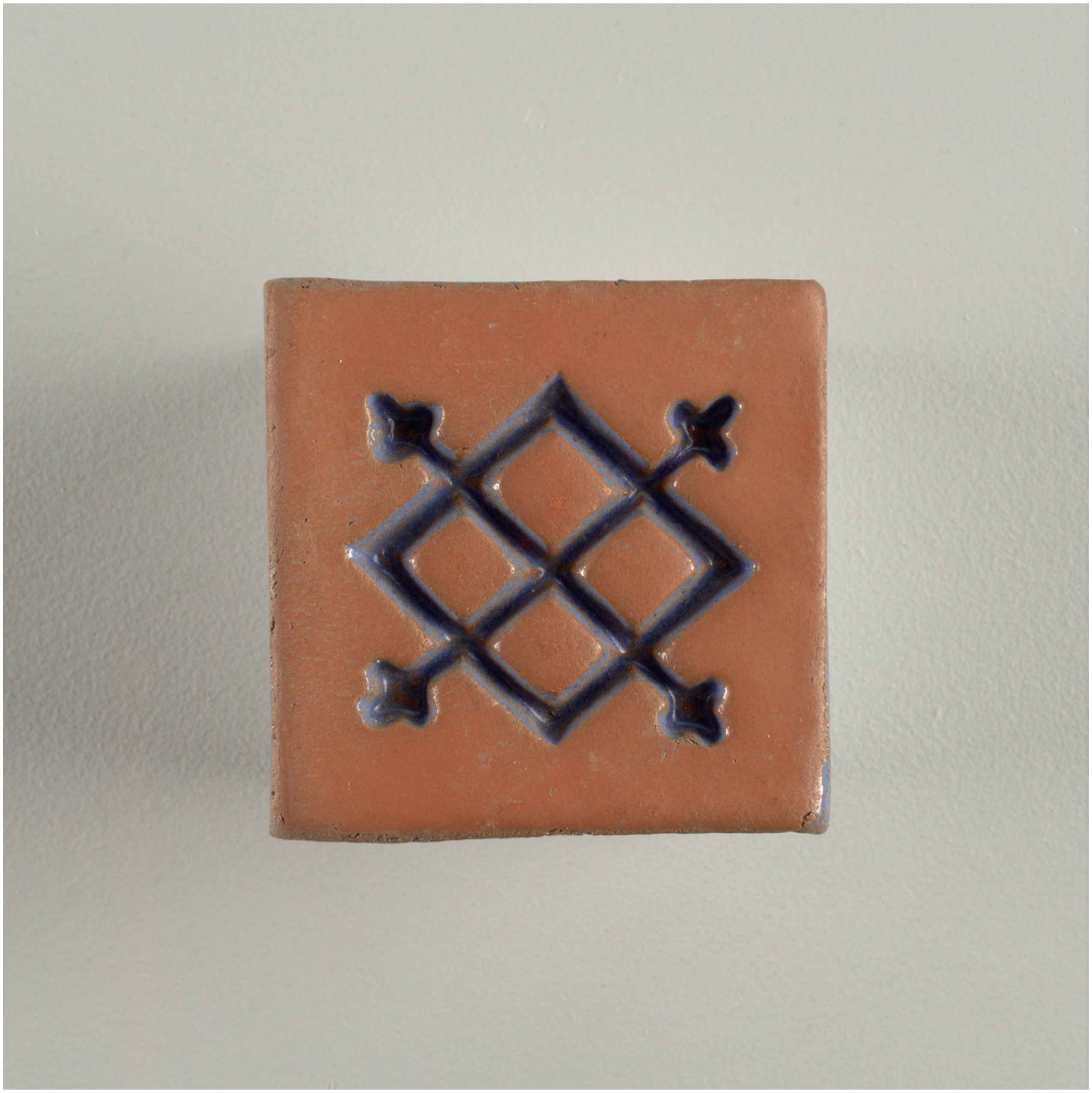
Max Gardner

Max is a summer 2017 intern on the Mapzen Mobility team. A PhD student at UC Berkeley, he will continue to support the Cavs even if LeBron leaves Cleveland again because, after all, Max left too.

© 2017 Mapzen

Get more for less with 512 pixel tiles

vector-tiles (/tag/vector-tiles) terrain-tiles (/tag/terrain-tiles) data (/tag/data) cartography (/tag/cartography) documentation (/tag/documentation) tangram (/tag/tangram) flex (/tag/flex)



(https://s3.amazonaws.com/mapzen-assets/images/512px-tiles/102111_1b02595b2be4f62c_b.jpg)

Starting today, Mapzen basemaps require fewer **Mapzen Flex (https://mapzen.com/pricing/)** requests (saving you money) and feature better label placement thanks to new 512 pixel vector and terrain tiles, and new versions of Tangram JS and ES.

We are also introducing buffered terrain tiles in both 260 and 516 tile sizes to enable new 3D and analytical applications.

Basemaps

Mapzen basemaps display and style data from the Mapzen vector and terrain tile services. Switching to 512 pixel size network requests can reduce overall latency and improve label placements in Tangram compared to 256 pixel tiles. Best of all, reduced network requests can also lower your **Mapzen Flex (https://mapzen.com/pricing/)** bill!

The following Mapzen basemap versions introduce 512 pixel tiles:

- Bubble Wrap: `8.0.0`
- Cinnabar: `8.0.0`
- Refill: `9.0.0`
- Tron: `5.0.0`
- Walkabout: `6.0.0`

These updates (along with other new features like **Mapzen icon library** support, **label steps** on all styles, and **Zinc graduating to a Refill theme**) are immediately available in **Mapzen.js** and via the **mapzen.com/carto (https://mapzen.com/documentation/cartography/styles/)** endpoints.

Tangram added 512 pixel tile support in JS **v0.13 (https://github.com/tangrams/tangram/releases/tag/v0.13.0)** and the ES **0.7 (https://github.com/tangrams/tangram-es/releases/tag/0.7.0)** releases. Android and iOS SDK support for updated basemap styles and 512 pixel tiles is coming soon.

512 pixel vector tiles

Historically, the first web slippy maps were based on 256 pixel-sized raster tiles. While vector tiles don’t store pixels, we still use the same handy `z/x/y` addressing scheme, and so our **v1.0 service (https://mapzen.com/blog/v1-vector-tile-service/)** launched with 256 pixel “logical” tiles last year.

Larger 512 pixel-sized tiles have grown more common in the industry and offer many benefits:

- **Fewer tiles, fewer network requests:** a single 512 request is equivalent to four 256 requests at all zooms, and a single zoom 15 request includes all data above zoom 16
- **Better labels:** map rendering tools like Tangram and MapboxGL get more room to place labels
- **Smaller overall file sizes:** a 512 tile compresses more efficiently than four 256 tiles
- **Offline:** Fewer 512 tiles are needed to cover the same geographic area, requiring less less disk space

Mapzen Flex (https://mapzen.com/pricing/) charges for API requests irrespective of tile size. A single 512 request is 25% the price of the four 256 requests for the equivalent “map view” of the same area, which means a lower Flex bill in most cases!

The new API endpoint pattern for vector tiles is:

```
https://tile.mapzen.com/mapzen/vector/v1/{tilesize}/{layers}/{z}/{x}/{y}.{format}?api_key
```

Note that 512 is opt-in. `{tilesize}/` is not required in the path and existing API requests will continue to function as before, returning the default size of 256.

The suggested max `{z}` value for 512 pixel tiles is zoom **15**. Like 256 pixel tiles, requesting `{z}` coordinates up to 20 will return a smaller geographic area, but the tile will not include any additional data already in the zoom 15 tile. (In practice, requesting 512 tiles up to zoom **16** can improve Tangram performance.)

To use the tiles in Tangram, specifying a **tile_size (https://mapzen.com/documentation/tangram/sources/#tile_size)** of 512 in the source block is strongly suggested, otherwise content will stretch and appear less detailed than intended.

Example Tangram YAML:

```
sources:
    mapzen:
        type: MVT
        url:  https://tile.mapzen.com/mapzen/vector/v1/512/all/{z}/{x}/{y}.mvt
        url_params:
            api_key: your-mapzen-api-key
        tile_size: 512
        max_zoom: 16
```

We've been beta-testing 512 pixel vector tiles since February, starting with the **v1.1 release (https://github.com/tilezen/vector-datasource/releases/tag/v1.1.0)**, which grouped neighbouring 2x2 chunks of tiles together for processing. (We call that a "metatile", although that name is often used to mean many different things!) This makes much better use of the map database by querying for a larger area and chopping the result up to make the original four tiles. We also reduce storage costs by keeping all four tiles together as a ZIP file that **Tapalcatl (https://github.com/tilezen/tapalcatl)** unpacks. This means a quicker API response, because unlike the database, we run Tapalcatl in many datacentres all over the world.

Terrain tiles

We announced the **Terrain Tiles service (https://mapzen.com/blog/terrain-tile-service/)** and the **public dataset (https://aws.amazon.com/blogs/publicsector/announcing-terrain-tiles-on-aws-a-qa-with-mapzen/)** just over one year ago in cooperation with the Amazon Web Services (AWS) team. The Terrain Tiles public dataset offers digital elevation models (DEMs) for the entire Earth, with varying resolutions of data sourced from open data providers from around the world. (We use the terrain data in both Walkabout and Tron to draw hillshading.)

To support larger 512 pixel vector tiles we've added a new microservice we call **Zaloa (https://github.com/tilezen/zaloa)** to stitch four 256 pixel tiles into one large 512 pixel tile on-the-fly, and introduce two new buffered sizes.

Both Terrarium and Normal formats now support tile sizes of 256 and 512 for basic map display, and buffered sizes of `260` and `516` pixels useful for 3D and analytical applications. When not provided, the size defaults to 256.

```
https://tile.mapzen.com/mapzen/terrain/v1/{tilesize}/{format}/{z}/{x}/{y}.{extension}?api
```

Besides the 512 pixel tile size benefits terrain tiles enjoy with vector tiles, new **buffered 260 and 516 pixel sized tiles offer several benefits:**

- **Fewer network requests:** The 2 pixel edge buffers reduce network requests from 9 to 1, as sampling neighboring tiles is no longer necessary. A single 516 request is equivalent to four 260 requests.
- **3D geometry construction**: The buffer provides enough height values in the overlap to calculate 3D elevation meshes without loading neighboring tiles.
- **Enables professional shading for 3D**: The 2nd pixel of buffer enables custom normals (slopes) to be calculated for the 1st pixel of the terrarium buffer without loading neighboring tiles.

The maximum `{z}` value for 512 pixel tiles is zoom **14**. Requesting `{z}` coordinates past that will result in a 404 error.

Example Tangram YAML:

```
sources:
    mapzen:
        type: MVT
        url:  https://tile.mapzen.com/mapzen/terrain/v1/512/normal/{z}/{x}/{y}.png
        url_params:
            api_key: your-mapzen-api-key
        tile_size: 512
        max_zoom: 14
```

Access to Mapzen’s terrain tiles continue to be free (with an API key) – they are not billed in Mapzen Flex.

Read the docs

Check out the full story in the project release notes:

- **Cartography (https://mapzen.com/documentation/cartography/get-started/)**
- Tangram JS **v0.13 (https://github.com/tangrams/tangram/releases/tag/v0.13.0)** and Tangram ES **0.7 (https://github.com/tangrams/tangram-es/releases/tag/0.7.0)**.
- **Vector tiles (https://mapzen.com/documentation/vector-tiles/use-service/#specify-tile-size)**

- **Terrain tiles (https://mapzen.com/documentation/terrain-tiles/use-service/#specify-tile-size)**
- **Mapzen.js (https://mapzen.com/documentation/mapzen-js/api-reference/#basemap-styles)**

Feedback

We're excited to see what you build and would love to hear feedback on the cartography or tiles. Please direct comments to **Nathaniel Vaughn Kelso (https://github.com/nvkelso)**, our Chief Cartographer, at **hello@mapzen.com (mailto:hello@mapzen.com)**!

*Photo credit via **Cooper Hewitt (https://collection.cooperhewitt.org/objects/18797885/)**, Tile; Gift of Florence Barnes; 1981-56-52.*

· 02 October 2017 ·

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

Matt Amos

OpenStreetMap developer-contributor and chilli enthusiast. Writes vector tile and other data-mastication tools at Mapzen.

Rob Marianski

Software engineer working on cutting tiles and infrastructure. Functional programming enthusiast.

© 2017 Mapzen

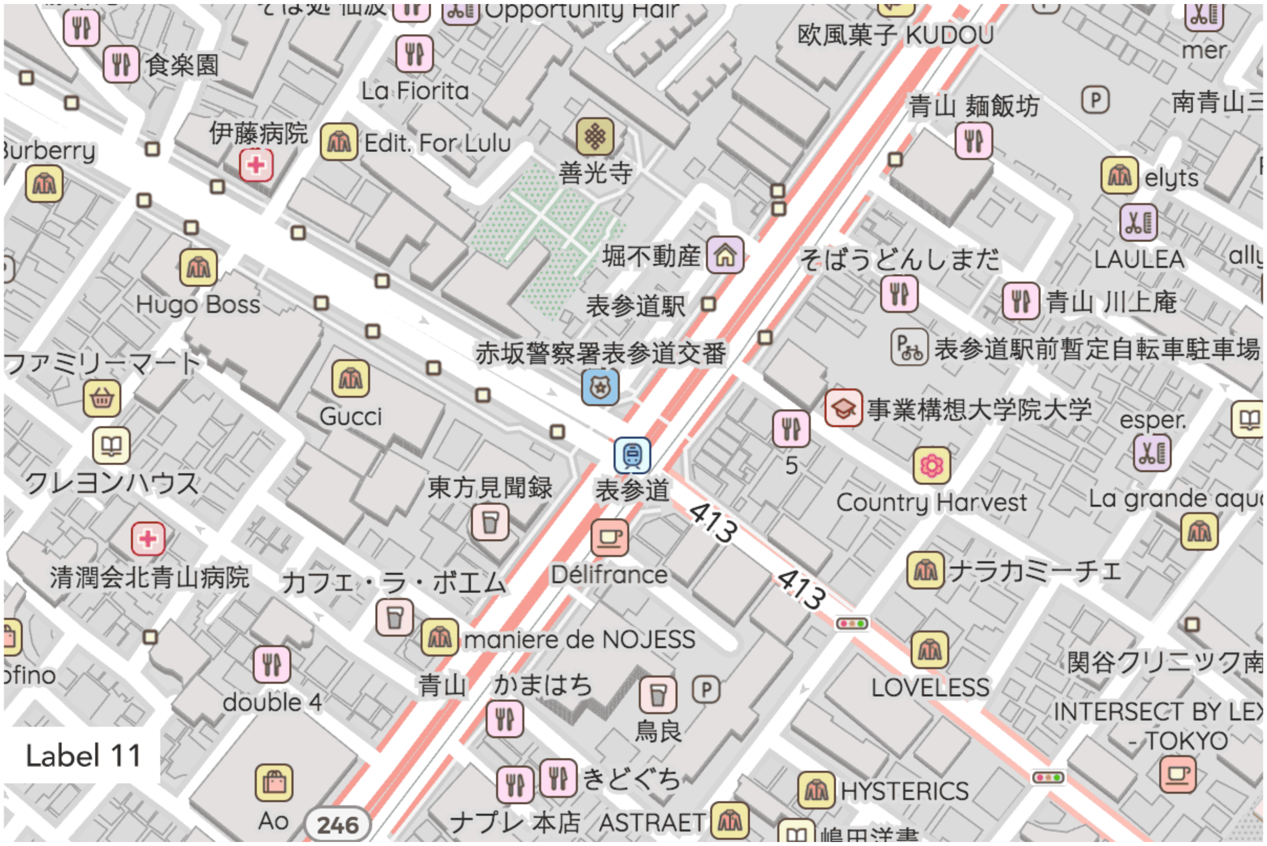
Mapzen Basemap Styles Updates

cartography (/tag/cartography) tangram (/tag/tangram)

All Mapzen Basemap styles now include the label levels **introduced in Refill (https://mapzen.com/blog/refill-levels-of-detail-and-labels/)** in July along with **512 pixel tile (https://mapzen.com/blog/512-pixel-tiles-and-basemaps)** support. Also, Zinc is graduating to a Refill theme and being retired a standalone basemap style.

Label Levels 0 to 11

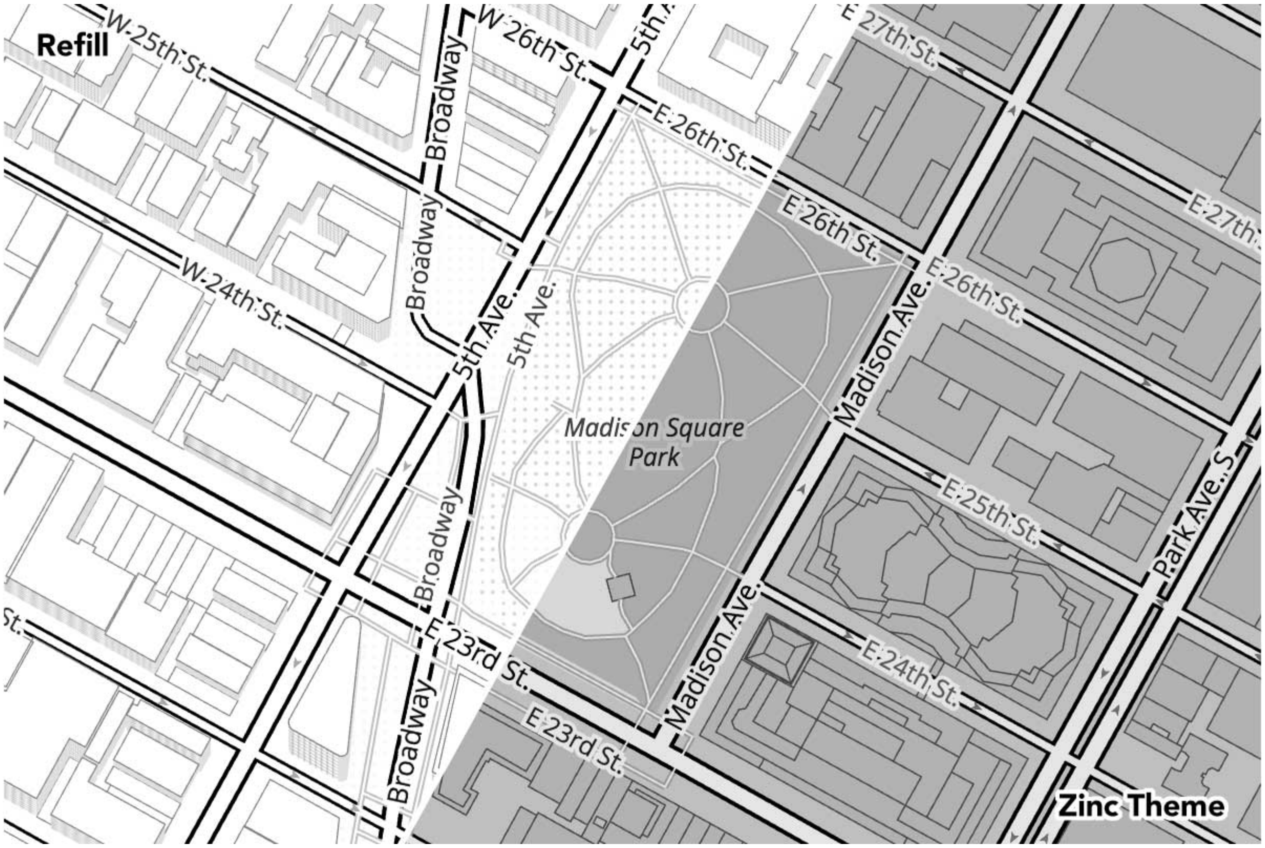
Label levels have been added to all Mapzen house styles! Bubble Wrap, Cinnabar, Tron and Walkabout now have label steps `0` to `11` , following the same label levels of Refill. See the **Levels of Detail (https://mapzen.com/blog/refill-levels-of-detail-and-labels/#label-density)** blog post for more info.


Opportunity Hair
食楽園
La Fiorita
欧風菓子 KUDOU
mer
青山 麺飯坊
南青山三
urberry
伊藤病院
Edit. For Lulu
善光寺
elyts
堀不動産
そばうどんしまだ
LAULEA
青山 川上庵
Hugo Boss
表参道駅
表参道駅前暫定自転車駐車場
ファミリーマート
赤坂警察署表参道交番
Gucci
事業構想大学院大学
esper.
5
クレヨンハウス
東方見聞録
表参道
413
Country Harvest
La grande aqu
清潤会北青山病院
カフェ・ラ・ポエム
Délifrance
413
ナラカミーチェ
maniere de NOJESS
関谷クリニック南
青山 かまはち
LOVELESS
fino
double 4
鳥良
INTERSECT BY LEX - TOKYO
Label 11
きどぐち
HYSTERICS
Ao
246
ナプレ 本店
ASTRAET

Bubble Wrap Label Levels - Omotesandō Avenue in Tokyo at zoom-17

Zinc and Refill

We’ve added Zinc as a new color theme to Refill. Zinc is unique among all the themes since it’s composed mostly of grays and removes all textures. Consider it a more flat and basic version of Refill.


Refill
W 25th St.
W 26th St.
5th Ave.
Broadway
E 27th St.
E 26th St.
W 24th St.
5th Ave.
Madison Square Park
Madison Ave.
E 25th St.
Park Ave. S
E 24th St.
E 23rd St.
E 27th St.
Zinc Theme

Importing Zinc

While Zinc has been deprecated as a standalone basemap style, it gets even more features as a Refill theme! Old versions are still published, but won’t be maintained – we recommend upgrading.

The example below imports Refill, adds Zinc as a color theme, and shows more labels.

Migrating to Label Levels

The following Mapzen basemap versions introduce Mapzen label themes.

```
import:
    - https://mapzen.com/carto/refill-style/9/refill-style.zip
    - https://mapzen.com/carto/refill-style/9/themes/color-zinc.zip
    - https://mapzen.com/carto/refill-style/9/themes/label-10.zip
```

- Bubble Wrap: `8.0.0`
- Cinnabar: `8.0.0`
- Tron: `5.0.0`
- Walkabout: `6.0.0`

Starting with the Mapzen basemap styles released today, you will be able to import the default `bubble-wrap-style` , `cinnabar-style` , `tron-style` , or `walkabout-style` basemap and then add or remove labels by importing a **label** theme. See below:

```
import:
    - https://mapzen.com/carto/bubble-wrap-style/8/bubble-wrap-style.zip
    - https://mapzen.com/carto/bubble-wrap-style/8/themes/label-10.zip
```

Mapping Label Variants to Label Levels

The `all-labels` and `no-labels` variants will still work in previous basemap versions, but going forward you'll need to upgrade and import the theme, specifying `label-0` or `label-10` to get equivalent label density. See the **documentation (https://mapzen.com/documentation/cartography/themes/)** for more information on migration.

· 02 October 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

Remixing the Mapzen Icon Library

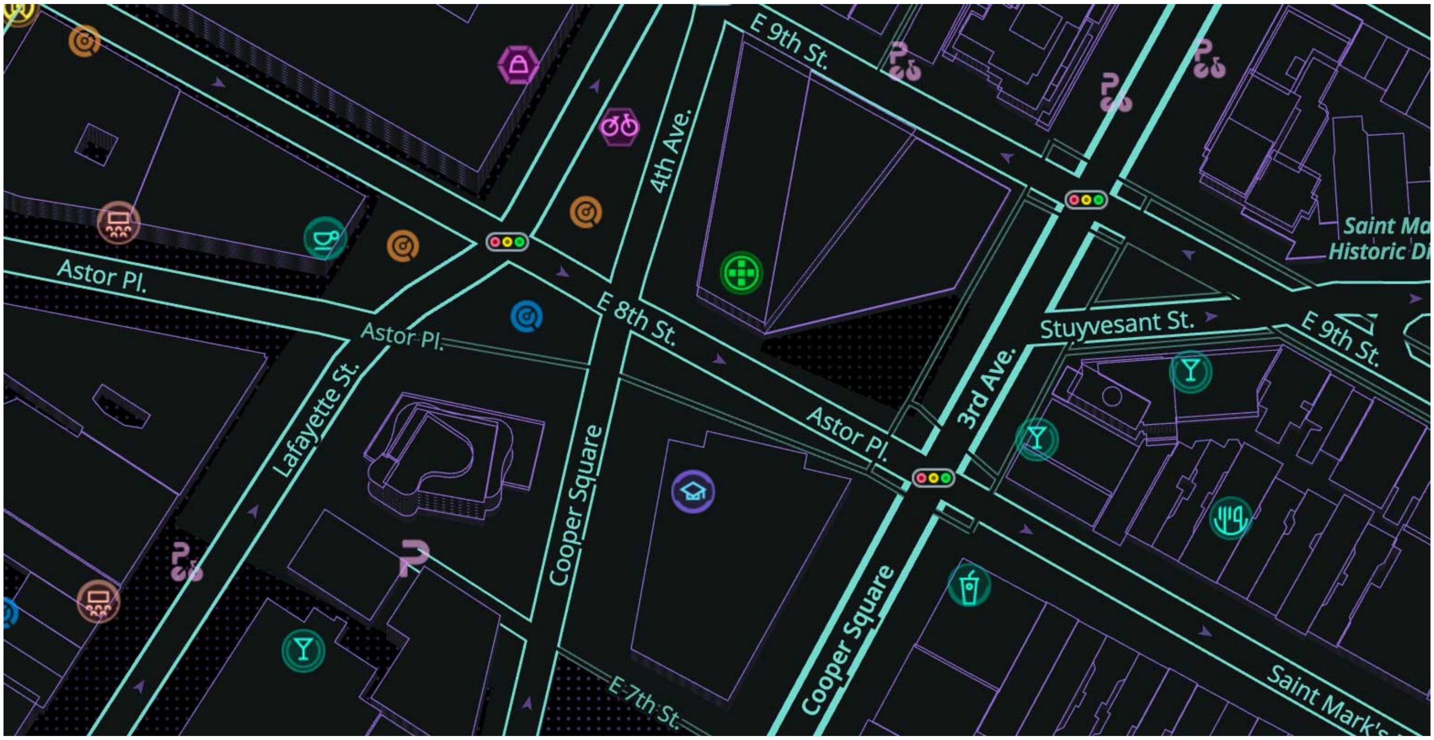
cartography (/tag/cartography) tangram (/tag/tangram)

We've updated the Mapzen Icon Library to allow you to import standalone Mapzen icons into your own custom Tangram map, and to remix icons between Mapzen basemap styles. Always wondered what Walkabout icons would look like on Refill? With our new icon imports that's easy. Or use our standalone PNG sprites to build in your favorite mapping library. Let's dive into how…

Mix and Match Icons and Mapzen Basemap Styles

Mix and match Mapzen icons with Mapzen basemap styles. It's simple! See the first example below, with Tron icons on Purple-Green Refill.

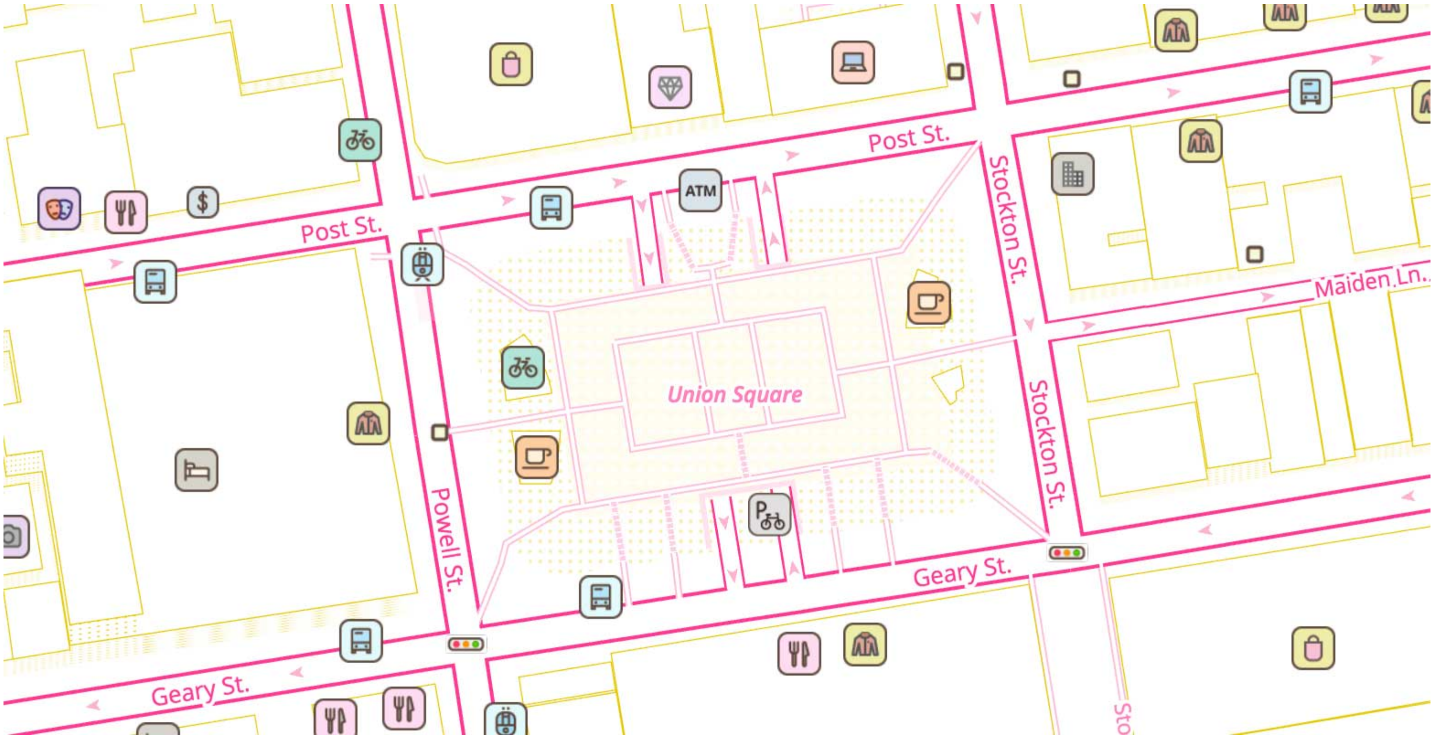
Tron Icons on Purple-Green Refill


E 9th St.
4th Ave.
Astor Pl.
Astor Pl.
E 8th St.
Lafayette St.
Cooper Square
Astor Pl.
3rd Ave.
Stuyvesant St.
E 9th St.
Saint Ma
Historic Di
Cooper Square
E 7th St.
Saint Mark's

In your Tangram scene, simply import the Refill house style then add the purple-green color theme and the label level of your choice. The last line imports Tron icons.

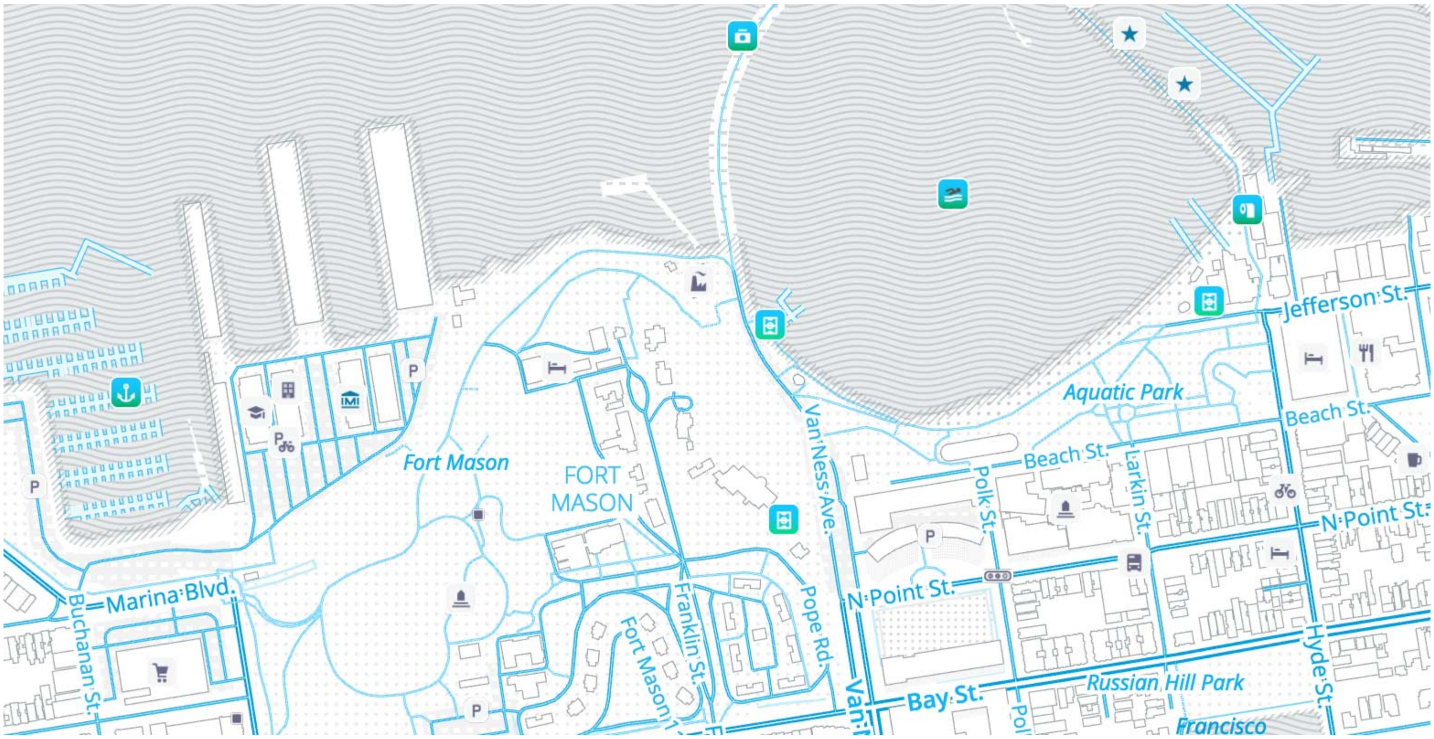
```
import:
    - https://mapzen.com/carto/refill-style/9/refill-style.zip
    - https://mapzen.com/carto/refill-style/9/themes/color-purple-green.zip
    - https://mapzen.com/carto/refill-style/9/themes/label-10.zip
    - https://mapzen.com/carto/tron-style/5/themes/tron-icons.zip
```

Bubble Wrap Icons on Pink-Yellow Refill


Post St.
Post St.
ATM
Stockton St.
Maiden Ln.
Union Square
Powell St.
Stockton St.
Geary St.
Geary St.
Sto

```
import:
    - https://mapzen.com/carto/refill-style/9/refill-style.zip
    - https://mapzen.com/carto/refill-style/9/themes/color-pink-yellow.zip
    - https://mapzen.com/carto/refill-style/9/themes/label-10.zip
    - https://mapzen.com/carto/bubble-wrap-style/8/themes/bubble-wrap-icons.zip
```

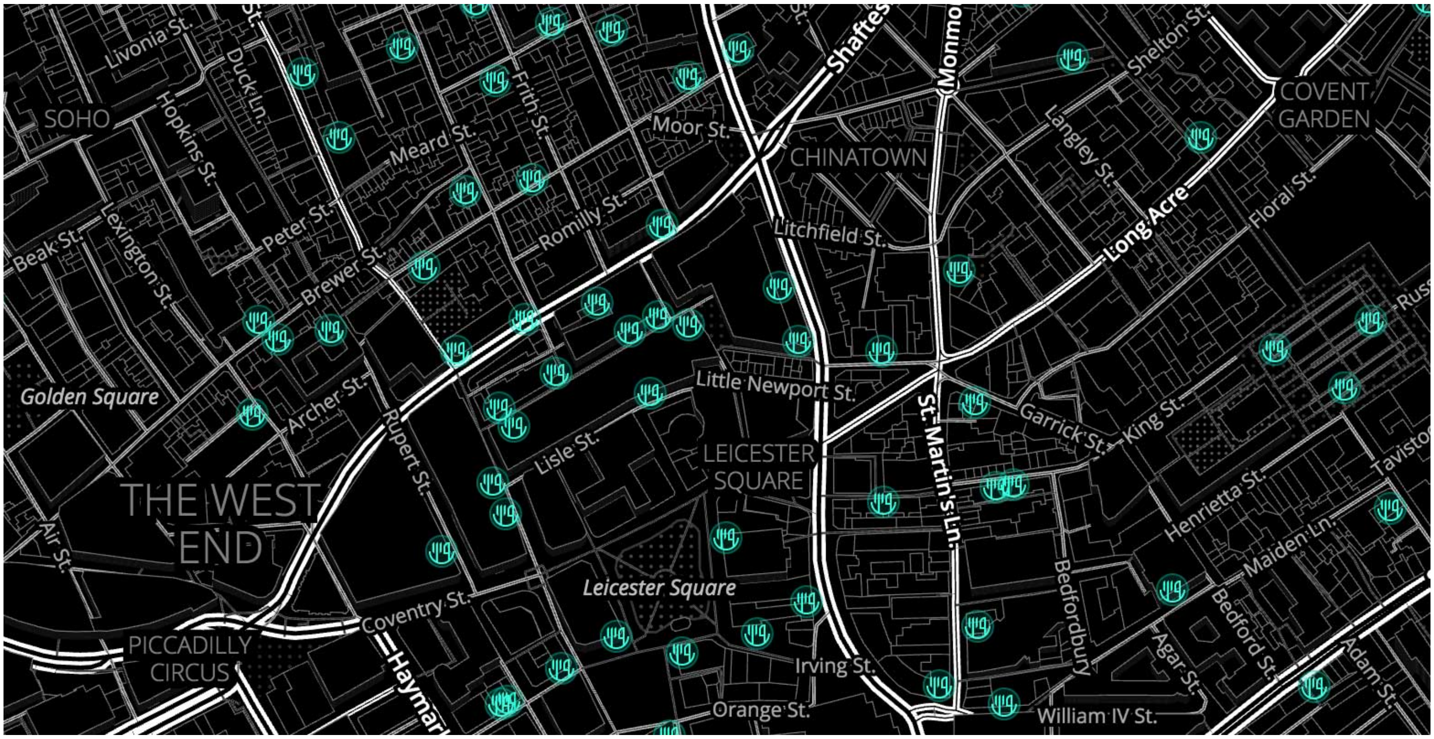
Walkabout Icons on Blue-Gray Refill


Jefferson St.
Aquatic Park
Beach St.
Beach St.
Fort Mason
FORT MASON
Van Ness Ave.
Polk St.
Larkin St.
N Point St.
N Point St.
Marina Blvd.
Buchanan St.
Franklin St.
Fort Mason
Pope Rd.
Bay St.
Russian Hill Park
Francisco
Hyde St.

```
import:
    - https://mapzen.com/carto/refill-style/9/refill-style.zip
    - https://mapzen.com/carto/refill-style/9/themes/color-blue-gray.zip
    - https://mapzen.com/carto/refill-style/9/themes/label-10.zip
    - https://mapzen.com/carto/walkabout-style/6/themes/walkabout-icons.zip
```

Add Icon by Name

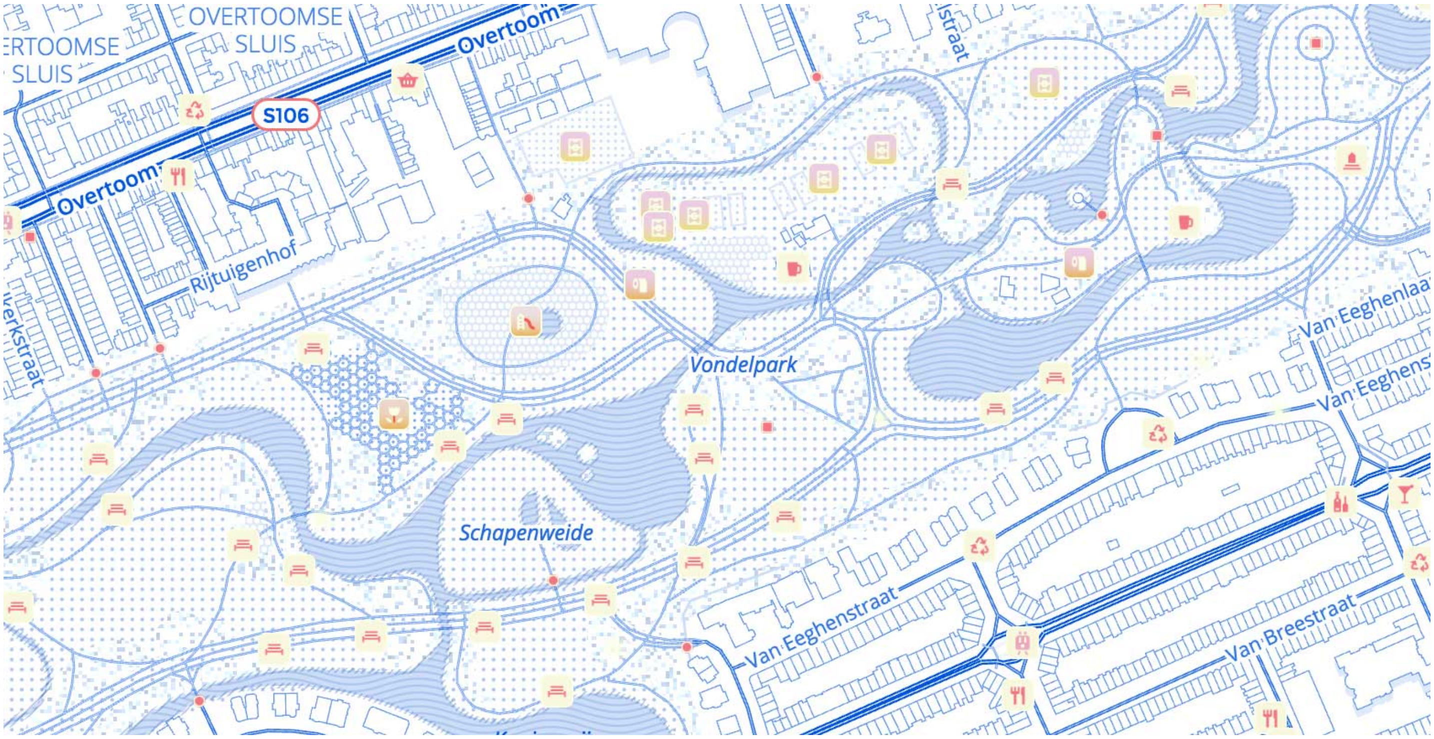
To add specific icons, add the name into the filter and call `mapzen_icon_library` when you draw. In this case, we’re filtering restaurants in London.


SOHO
Livonia St.
Duck Ln.
Hopkins St.
Meard St.
Frith St.
Moor St.
Shaftes
Monmo
Shelton St.
COVENT GARDEN
CHINATOWN
Langley St.
Floral St.
Long Acre
Litchfield St.
Romilly St.
Peter St.
Beak St.
Lexington St.
Brewer St.
Golden Square
Archer St.
Rupert St.
Little Newport St.
St. Martin's Ln.
Garrick St.
King St.
Lisle St.
LEICESTER SQUARE
THE WEST END
Air St.
Henrietta St.
Maiden Ln.
Tavisto
Russ
Leicester Square
Coventry St.
Bedfordbury
Agar St.
Bedford St.
Adam St.
PICCADILLY CIRCUS
Haymark
Irving St.
Orange St.
William IV St.

```
import:
    - https://mapzen.com/carto/refill-style/9/refill-style.zip
    - https://mapzen.com/carto/refill-style/9/themes/color-inverted.zip
    - https://mapzen.com/carto/tron-style/5/themes/tron-icons.zip

layers:
    my_pois:
        data: { source: mapzen, layer: pois }
        filter: { kind: restaurant }
        draw:
            mapzen_icon_library:
                text:
                    visible: false
```

Customize Icon Colors

Customize icon colors like in the example below. Here, Walkabout icons are shaded yellow and pink. Specify the color of your choice within the shader `colorized_icons` in `u_tint` .


OVERTOOMSE SLUIS
ERTOOMSE SLUIS
Overtoom
S106
Overtoom
Rijtuigenhof
erkstraat
straat
Vondelpark
Schapenweide
Van Eeghenlaan
Van Eeghenstraat
Van Eeghenstraat
Van Breestraat

```
import:
    - https://mapzen.com/carto/refill-style/9/refill-style.zip
    - https://mapzen.com/carto/refill-style/9/themes/label-11.zip
    - https://mapzen.com/carto/walkabout-style/6/themes/walkabout-icons.zip
    - https://mapzen.com/carto/refill-style/9/themes/color-blue.zip

styles:
    colorized_icons:
        shaders:
            uniforms:
                u_tint: [0.925,0.090,0.094]
```

Use Icons Independently

In Tangram

Create your own custom map and use Mapzen icons. This example uses Tron icons on a custom map of building footprints drawn with blue lines in a Tangram scene.

Sprites on their own

The Mapzen Icon Library is available for use in your favorite mapping or graphics software as individual sprite images. (Currently only `png` format at `2x` resolution is supported.)

- `https://mapzen.com/carto/bubble-wrap-style/8/icons/{resolution}/{sprite}.{format}`
- `https://mapzen.com/carto/cinnabar-style/8/icons/{resolution}/{sprite}.{format}`
- `https://mapzen.com/carto/refill-style/9/icons/{resolution}/{sprite}.{format}`
- `https://mapzen.com/carto/tron-style/5/icons/{resolution}/{sprite}.{format}`
- `https://mapzen.com/carto/walkabout-style/6/icons/{resolution}/{sprite}.{format}`
- `https://mapzen.com/carto/sdk-default-style/1/icons/{resolution}/{sprite}.{format}`

For example:

Load the `adit` sprite in the **Bubble Wrap** style at `2x` in `png` format:

```
https://mapzen.com/carto/bubble-wrap-style/8/icons/2x/adit.png
```

```
import:
    - https://mapzen.com/carto/tron-style/5/themes/tron-icons.zip

sources:
    mapzen:
        type: TopoJSON
        url: https://tile.mapzen.com/mapzen/vector/v1/all/{z}/{x}/{y}.topojson

layers:
    my_earth:
        data: { source: mapzen, layer: earth }
        draw:
            polygons:
                order: 0
                color: [0.000, 0.000, 0.000, 1.00]

    my_buildings:
        data: { source: mapzen, layer: buildings }
        filter: { $zoom: { min: 13 } }
        draw:
            lines:
                order: 4
                width: 2px
                color: [0.192, 0.118, 0.612, 1.00]

    my_pois:
        data: { source: mapzen, layer: pois }
        draw:
            mapzen_icon_library:
                text:
                    visible: false
```

See our **Icon Library documentation (https://mapzen.com/documentation/cartography/icons/)** for more information, and **let us know what you make (https://twitter.com/intent/tweet?text=I+remixed+@mapzen+icons!)**!

· 03 October 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

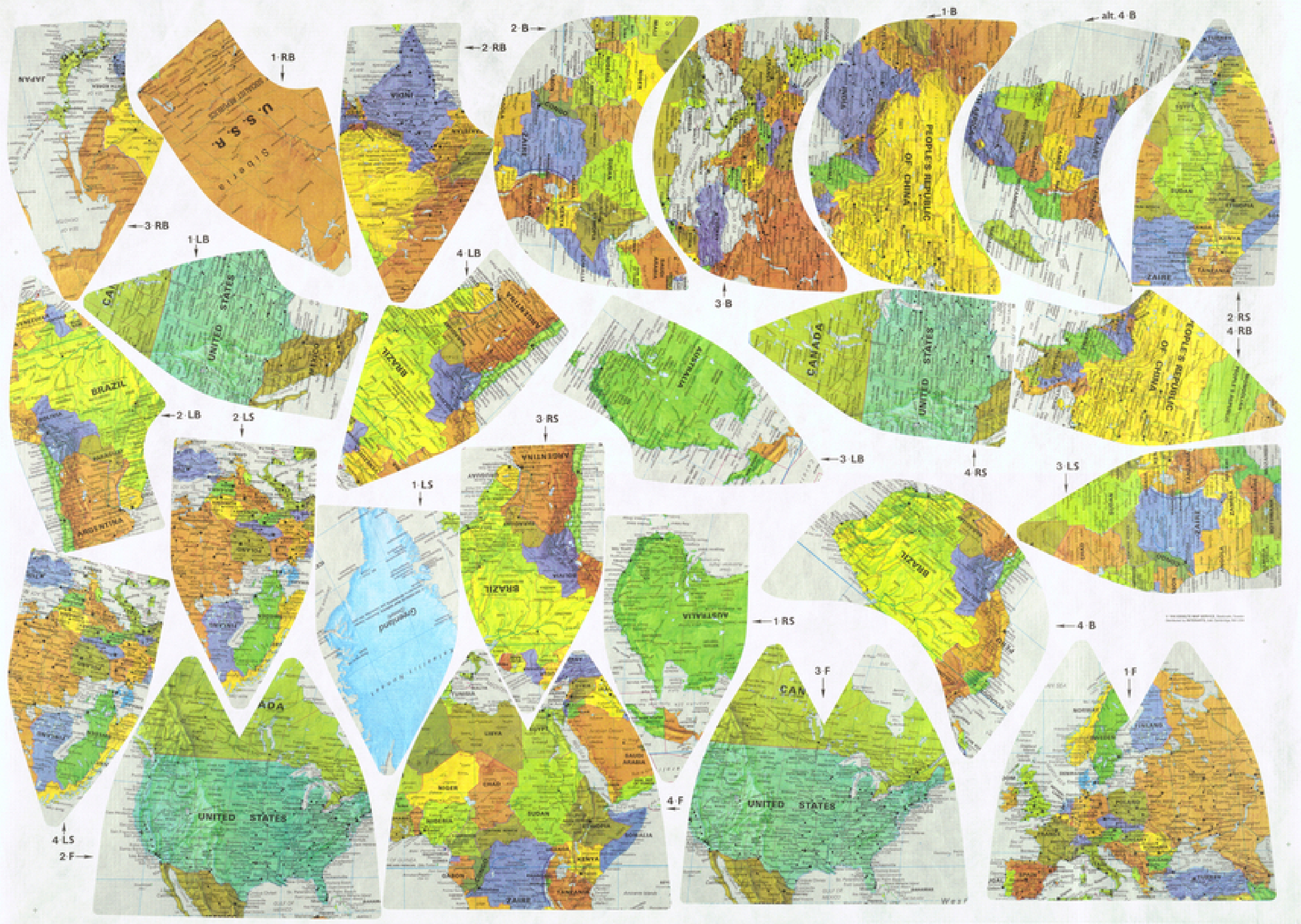
Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

Mapzen Places is here! And there! And everywhere.

places (/tag/places) flex (/tag/flex) data (/tag/data) whosonfirst (/tag/whosonfirst)


1·RB
U.S.S.R.
Siberia
JAPAN
INDIA
2·B
2·RB
1·B
alt. 4·B
PEOPLE'S REPUBLIC OF CHINA
3·RB
1·LB
4·LB
3·B
2·RS
4·RB
CANADA
UNITED STATES
BRAZIL
AUSTRALIA
ARGENTINA
2·LB
2·LS
3·RS
3·LB
4·RS
3·LS
1·LS
Greenland
1·RS
4·B
3·F
1·F
4·F
4·LS
2·F
ZAIRE
SUDAN
ETHIOPIA
KENYA

Mapzen Places (https://mapzen.com/data/places/) is here! Find out more about a location, its geometry, its relationships, and return useful metadata, all through a **convenient and well-documented API (https://mapzen.com/documentation/places/)**. Mapzen Places is all based on open data, with licensing suitable for commercial projects. Start using it via **Mapzen Flex (https://mapzen.com/pricing/)** today!

If you like plain-text GeoJSON records with stable numeric IDs, a finite set of placetypes, and a common set of ancestors for all records, then Mapzen Places is for you. Perform spatial queries without having to set up and run a spatial database! Get worldwide admin hierarchies from continent to neighbourhood! Records may be updated, superseded or deprecated, but never removed. Mapzen Places also "holds hands" with Wikipedia, Wikidata and other datasets.

The Mapzen Places API gets its data from **Who's On First (https://whosonfirst.mapzen.com/)**, and supersedes the Who's On First API which we have been beta testing for the past year. Note that `whosonfirst-api.mapzen.com/?` is deprecated but requests will get redirected to `mapzen.places.com/v1/?`. The old `whosonfirst.*` methods will continue to work but are deprecated — `mapzen.*` is the new method.

There are **many useful methods in the Mapzen Places API (https://mapzen.com/documentation/places/methods/)**, including

mapzen.places.getDescendants

Get descendants of a place (https://mapzen.com/documentation/places/methods/#mapzen.places.getDescendants): counties in a country, postal codes in a city, regions in a continent, venues in a neighbourhood.

Here's the API request for neighbourhoods in London:

```
https://places.mapzen.com/v1/?
method=mapzen.places.getDescendants&
id=101750367&
placetype=neighbourhood&
extras=name
&api_key=INSERT-YOUR-API-KEY
```

(We've broken out the arguments in the URL to make it easier to read – ***here's a link (https://places.mapzen.com/v1/?method=mapzen.places.getDescendants&id=101750367&placetype=neighbourhood&extras=name&api_key=INSERT-YOUR-API-KEY)****, just add your API key.)*

Here's the response:

```
places: [
    {
        "wof:id": "85785187",
        "wof:parent_id": "101750367",
        "wof:name": "Abbey Wood",
        "wof:placetype": "neighbourhood",
        "wof:country": "GB",
        "wof:repo": "whosonfirst-data",
        "name": ""
    },
    {
        "wof:id": "85785221",
        "wof:parent_id": "101750367",
        "wof:name": "Acton",
        "wof:placetype": "neighbourhood",
        "wof:country": "GB",
        "wof:repo": "whosonfirst-data",
        "name": ""
    }...
```

Adding `format=geojson` does what you'd expect:

```
{
type: "FeatureCollection",
features: [
    {
        type: "Feature",
        geometry: {
            type: "Point",
            coordinates: [
                0.12401,
                51.495463
            ]
        },
        properties: {
            wof<img class='emoji' title='id' alt='id' src='https://github.global.ssl.fast
            wof:parent_id: "101750367",
            wof:name: "Abbey Wood",
            wof:placetype: "neighbourhood",
            wof:country: "GB",
            wof:repo: "whosonfirst-data",
            name: "",
            geom:latitude: 51.495463,
            geom:longitude: 0.12401,
            lbl:latitude: 51.488981,
            lbl:longitude: 0.11348
     }
 }...
```

...as does `format=csv` :

```
name,wof_country,wof_id,wof_name,wof_parent_id,wof_placetype,wof_repo
,GB,85785187,"Abbey Wood",101750367,neighbourhood,whosonfirst-data
,GB,85785221,Acton,101750367,neighbourhood,whosonfirst-data
,GB,85785229,Addiscombe,101750367,neighbourhood,whosonfirst-data
,GB,85785259,Aldersbook,101750367,neighbourhood,whosonfirst-data
...
```

mapzen.places.getHierarchiesByLatLon

Get the **hierarchy of a place (https://mapzen.com/documentation/places/methods/#mapzenplacesgethierarchiesbylatlon)**, from `neighbourhood` or `microhood` up to `country` and `continent` . Here is the hierarchy for a microhood in Bernal Heights, San Francisco:

```
https://places.mapzen.com/v1/?
method=mapzen.places.getHierarchiesByLatLon&
latitude=37.74
&longitude=-122.42&
api_key=INSERT-YOUR-API-KEY
```

```
{
hierarchies: [
        {
            microhood_id: "420780693",
            neighbourhood_id: 85865945,
            continent_id: 102191575,
            country_id: 85633793,
            locality_id: 85922583,
            county_id: 102087579,
            region_id: 85688637
        }
    ],
stat: "ok"
}
```

Adding `spr=1&extras=name&` will give you a more detailed response – here's a snippet of the `neighbourhood` details:

```
    neighbourhood: {
        "wof:id": 85865945,
        "wof:parent_id": "85922583",
        "wof:name": "Bernal Heights",
        "wof:placetype": "neighbourhood",
        "wof:country": "US",
        "wof:repo": "whosonfirst-data"
    }...
```

mapzen.places.getIntersects

Discover **placetypes that intersect a bounding box (https://mapzen.com/documentation/places/methods/#mapzen.places.getIntersects)** – here is the request for counties in `-6.6482,38.2448,-4.8118,39.9779` :

```
https://places.mapzen.com/v1/?
method=mapzen.places.getIntersects&
min_latitude=-6.6482&
min_longitude=38.2448&
max_latitude=-4.8118&
max_longitude=39.9779&
placetype=county&
api_key=INSERT-YOUR-API-KEY
```

a.k.a. coastal Tanzania:

```
{
  "places": [
    {
      "wof:id": 421188197,
      "wof:parent_id": 85679651,
      "wof:name": "Kusini",
      "wof:placetype": "county",
      "wof:country": "TZ",
      "wof:repo": "whosonfirst-data"
    },
    {
      "wof:id": 421191173,
      "wof:parent_id": 85679651,
      "wof:name": "Kati",
      "wof:placetype": "county",
      "wof:country": "TZ",
      "wof:repo": "whosonfirst-data"
    }...
```

These methods have **lots of arguments (https://mapzen.com/documentation/places/methods/#arguments_8)**, and as noted earlier, can return data in exciting formats such as json, geojson, and csv.

If you choose GeoJSON, you can add the API response directly onto a map! In Tangram, here's how you can use `mapzen.places.getDescendants` to display airports in Africa:

```
    venues:
        type: GeoJSON
        # url: https://whosonfirst-api.mapzen.com/?method=whosonfirst.places.getNearby
        url: https://places.mapzen.com/v1/?
        url_params:
            method: mapzen.places.getDescendants
            id: 102191573 #africa
            api_key: global.sdk_mapzen_api_key
            placetype: campus
            tags: airport
            format: geojson
            per_page: 500
            extras: wof:tags,wof:concordances,mz:is_current,name:eng_x_variant,oa:type,oa
```

Note the `extras` — this lets you pull in metadata (in this case, airport codes and airport size that we've concorded with OurAirports and ICAO and IATA) that you can use for labels and display.

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Open full screen (https://mapzen.com/tangram/view/?api=22/1036#5.605/5.148/3.729), and **check out the scene file (https://mapzen.com/tangram/play/?api=22/1036#5.605/5.148/3.729)** in Tangram Play!

You can learn much more on the **Mapzen Places page (https://mapzen.com/data/places/)** and **the documentation (https://mapzen.com/documentation/places/)**. Mapzen Places is part of **Mapzen Flex (https://mapzen.com/pricing/)**, with clear, predictable pricing and a generous free tier. Sign up for **an API key (https://mapzen.com/developers)** to get started!

Preview image: Sheet printed with parts of map of the world in sections as would be made up into baseball hats. The parts of 4 hats are printed on each sheet. It was designed by Esselte Map Service and company: Interarts and made by Graphic Jackets, via ***Cooper Hewitt (https://collection.cooperhewitt.org/objects/18635637/)****.*

· 05 October 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

Analyzing historical transit service & GTFS publishing practices in Transitland

mobility (/tag/mobility) transitland (/tag/transitland) interns (/tag/interns)

The original publication of the **General Transit Feed Specification (https://developers.google.com/transit/gtfs/)** (GTFS) in 2006 massively increased the availability and accessibility of transit data, and enabled countless new applications of this data for routing, spatial analysis, and city planning. A decade later, the historic collection of published feeds is also a valuable resource for understanding how transit systems evolve over time, and how service and investment decisions can shape our transportation choices and the built environment.

GTFS Data Exchange (http://www.gtfs-data-exchange.com/), a project started in 2008 by **Jehiah Czebotar (https://twitter.com/jehiah)** with just a dozen feeds, was one of the first compilations of public feeds and archived data, and eventually grew to approximately 1,000 feeds, with over 13,000 archived feed versions totaling 93 gigabytes.

GTFS Data Exchange entered read-only mode in 2016, with **Transitland becoming one of the services to take its place as an ever-updating archive of feeds (https://transit.land/news/2016/05/16/gtfs-data-exchange.html)**. This summer I began a project to import some of GTFS Data Exchange's historical feeds versions that predate Transitland into the **Transitland Feed Registry (https://transit.land/feed-registry/)**, and to use this combined data to better understand how often feed updates are published, the amount of time covered by each schedule, and trends in how much transit service is provided by each operator.

Retrieving information from GTFS Data Exchange

Since GTFS Data Exchange was built from community contributed data, the quality of the data and the naming conventions was unstandardized. While some GTFS Data Exchange sources were easily matched with Transitland Feeds using simple searches (e.g. Caltrain has no name variants), others were unsearchable or duplicated in the database due to alternative names or acronyms (Bay Area Rapid Transit shortened to BART, for one).

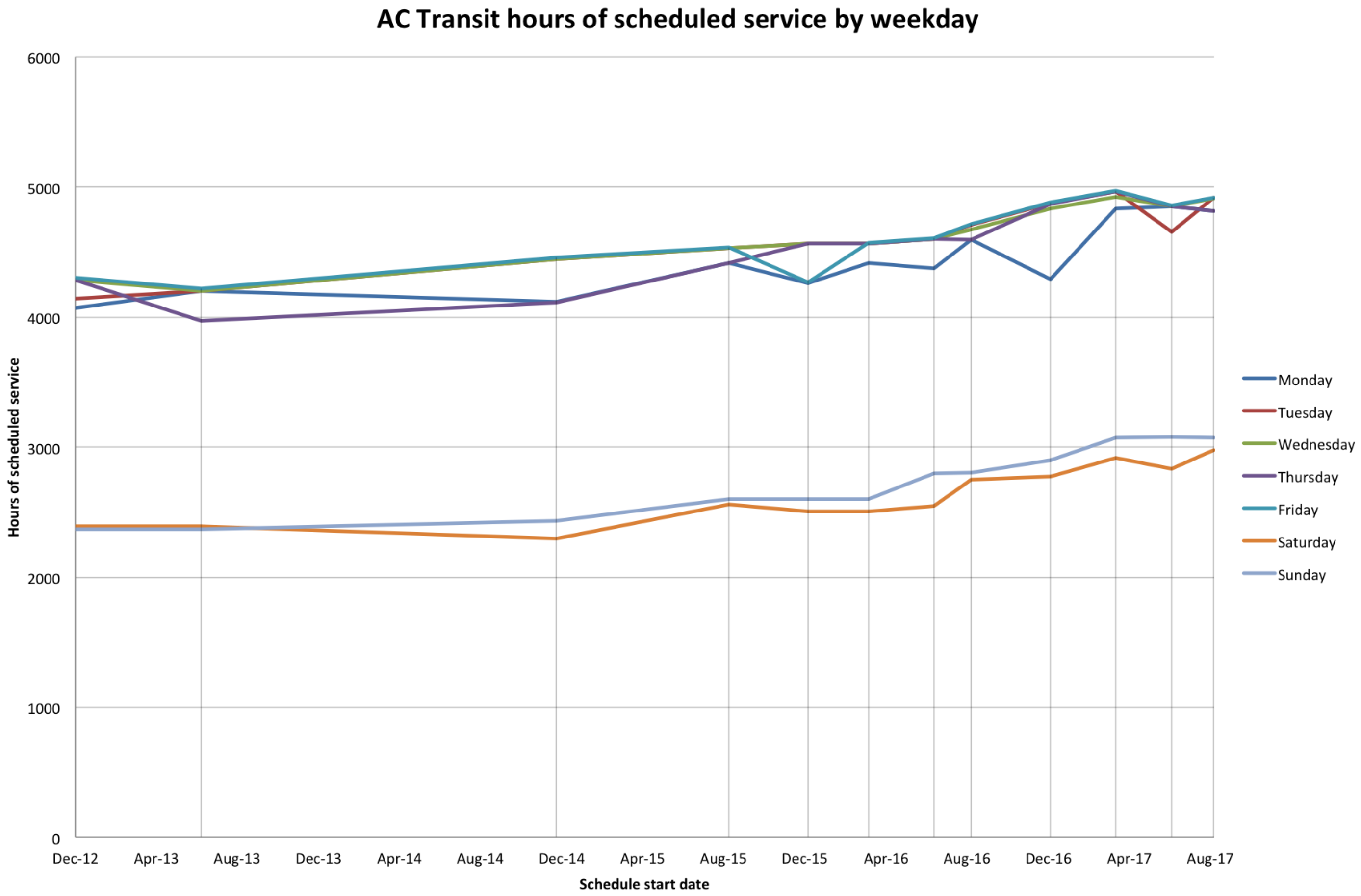
In order to address this, I developed two conventions for identifying matches between GTFS Data Exchange and Transitland Feeds:

1. A name search that identifies exact name matches
2. Source match: where the feed URI for both sources matched exactly

This semi-automatic matching associated **311 out of 845 Transitland Feeds (https://transit.land/dispatcher/feeds?tag_key=gtfs_data_exchange)** with their equivalent in GTFS Data Exchange, allowing me to upload about 3,000 historic feed versions that predated the creation of Transitland. We welcome others to **help us manually associate additional feeds (https://github.com/transitland/gtfs-data-exchange-archiver/issues/3)**.

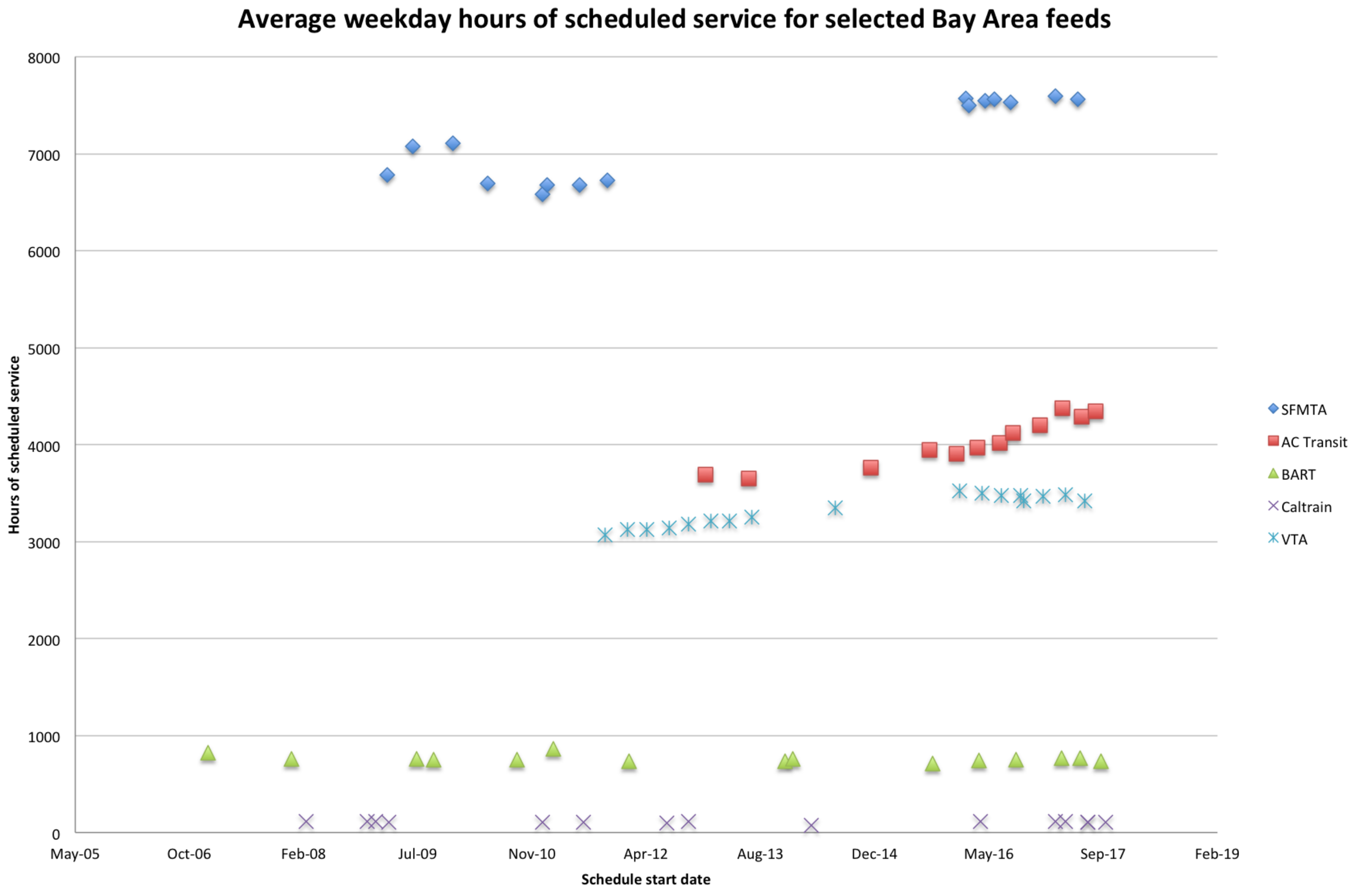
Historic levels of service

Whenever a new feed version is added to Transitland, several statistics are generated to help understand the contents. One way the feed version is characterized is to calculate the number of **service hours (https://mapzen.com/blog/transitland-feed-reports/)** for every day in the schedule. The addition of the historic data allows us to compare at the levels of service from several years ago to the present day.



This example from AC Transit, using feed versions from 2012-onwards, highlights a slow but steady restoration of service cuts made during the Great Recession, particularly in weekend service.

Comparing service hours over time in between different transit feeds also helps us understand the larger picture of the entire region. Policy changes could have shifted the level of service in an entire area, or budget realignments may have helped support the operating costs of buses, but not necessarily rails or ferries. For these use cases, I’ve aggregated the information provided above for each feed into an average service hour per feed version. Below is a graph including the Caltrain, BART, VTA, and AC Transit feeds.



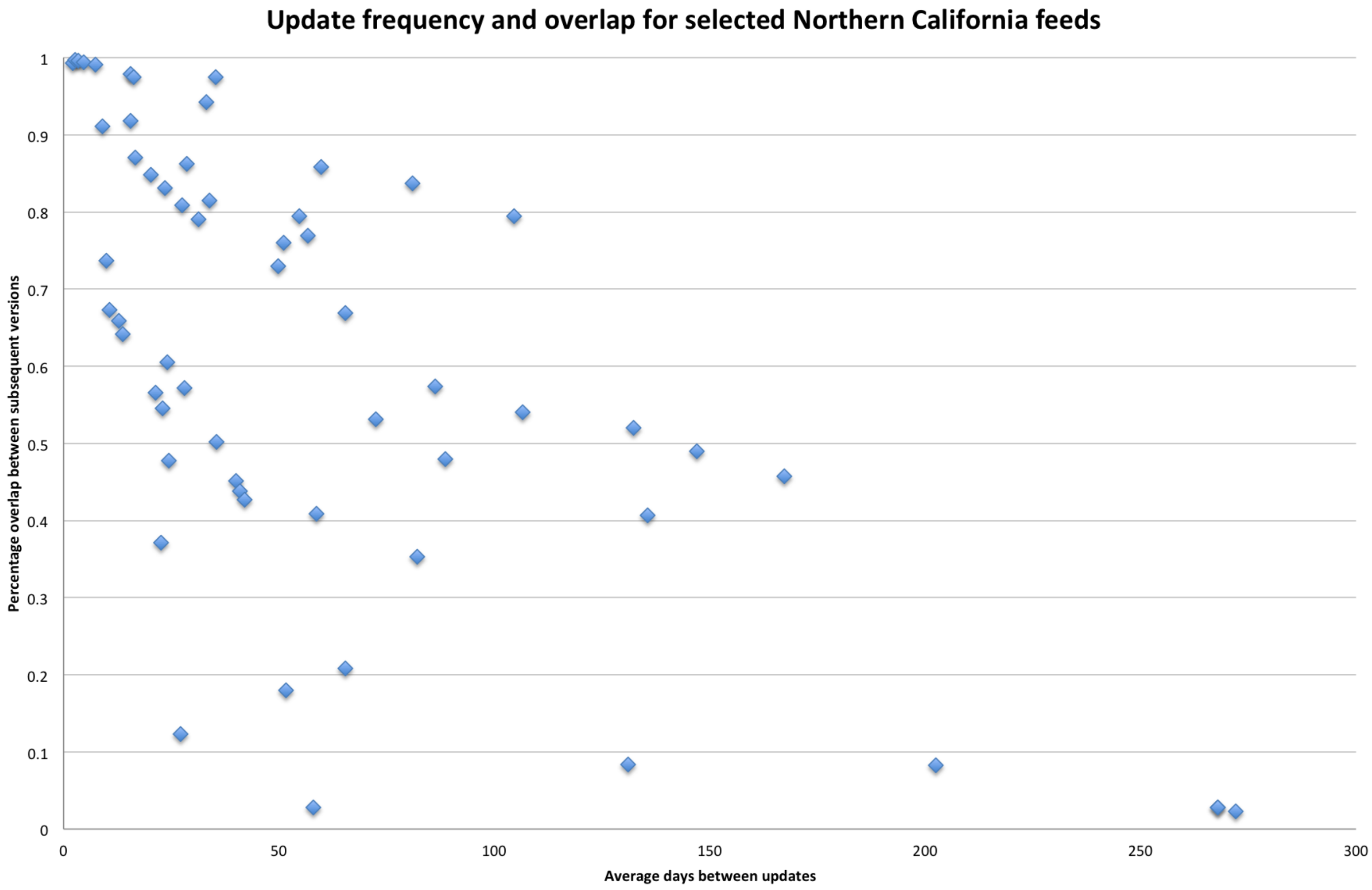
Although there's a large difference in the level of service hours between feeds, it's important to recognize that routes can be vastly different and therefore it's more valuable to compare relative, rather than absolute change in service hours. Here, we can see that since their first feed versions, AC Transit, Valley Transportation Authority (VTA), and San Francisco Municipal Transportation Authority (SFMTA) have all increased their levels of service, while BART and Caltrain's service times have remained relatively level.

While it might be straightforward to make assumptions that more service means better service, that's not necessarily the case. While the Caltrain and BART are regional rail systems that move their passengers long distances across the entire Bay Area, AC Transit, VTA, and SFMTA are all bus systems which prioritize coverage and accessibility within their particular cities of concern. The operating range and to an extent, frequency of the BART and Caltrain relies on a heavy investment in rail, whereas bus systems are more far flexible in experimenting with their routes and schedules.

Feed update frequency and duration

Another statistic this series of graphs revealed is the variation in frequency of feed version updates, which led to a few more exploratory questions. As we can see in Average Service Times in the Bay Area, some feeds update their feed versions more often than others. Does this mean we're always using up-to-date information in these other feeds? How does the duration of each of these feed versions relate to the frequency of updates?

Transitland fetches the URL of each feed every day to check for new versions. Most providers publish updated feeds on a regular schedule, usually every few weeks to every few months, and follow the **GTFS best practice (http://gtfs.org/best-practices/#publishing)** of publishing a new feed before the current version has less than 1 week of service left. However, sometimes the current feed version ends on or before the start date of the next feed version — in other words, we're using out-of-date information. In these cases, Transitland attempts to "fill in the gap" by extending the scheduled service in the current feed version, but this isn't always accurate — they are, after all, just predictions. In order to represent updated information, we want at least a small overlap in the service period of each feed version.

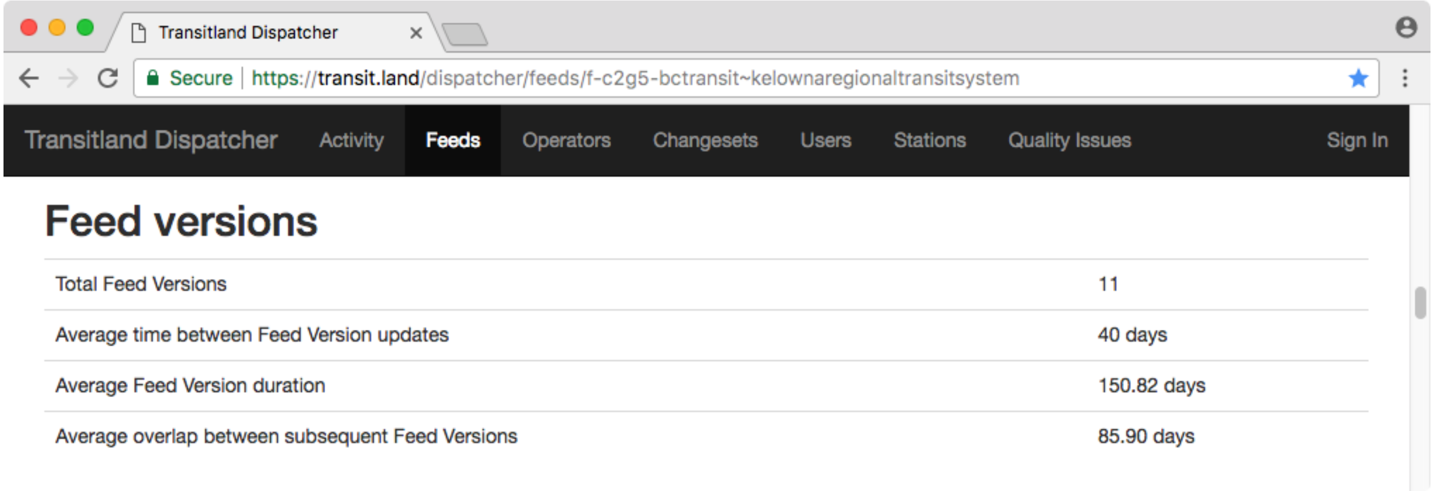


Here, for each feed in Northern California, we plot the average number of days between subsequent feed versions being published, and the percentage of schedule overlap (defined as the number of days shared in common as a percentage of the total time period covered by both versions).

We can see that frequently published feeds (as often as every day) tend to have a high degree of schedule overlap, whereas the least frequently published feeds (once or twice per year) tend to have less overlap. This suggests that feed consumers should take additional care with infrequently published feeds to prevent a lapse in scheduled service.

Update statistics in the Transitland API

Statistics on feed updates can now be generated for each Feed using the **Transitland API (https://transit.land/documentation/datastore/feeds.html)**, and are also shown in the **Dispatcher page (https://transit.land/dispatcher/feeds)** for each Feed.


Transitland Dispatcher
Secure | https://transit.land/dispatcher/feeds/f-c2g5-bctransit~kelownaregionaltransitsystem
Transitland Dispatcher
Activity
Feeds
Operators
Changesets
Users
Stations
Quality Issues
Sign In

Feed versions

Total Feed Versions	11
Average time between Feed Version updates	40 days
Average Feed Version duration	150.82 days
Average overlap between subsequent Feed Versions	85.90 days

(https://transit.land/dispatcher/feeds/f-c2g5-bctransit~kelownaregionaltransitsystem)

Conclusion

Our hope is that the availability of archival feed versions — some as early as 2008 — will enable researchers to answer questions about how transit systems change over time. We plan to continue adding more historic feed versions to Transitland and documenting this process on **the GitHub repository for this project (https://github.com/transitland/gtfs-data-exchange-archiver)**. We also encourage interested users to explore the archive and let us know what you discover. Finally, if you have historical GTFS feeds to contribute, please do **drop us a line (mailto:hello@transit.land)**.

· 10 October 2017 ·

Christine Phan

Chris is a CS undergraduate at Stanford and was a 2017 Mapzen Mobility intern. She is interested in measuring spatial inequality through access to jobs, housing, and transit.

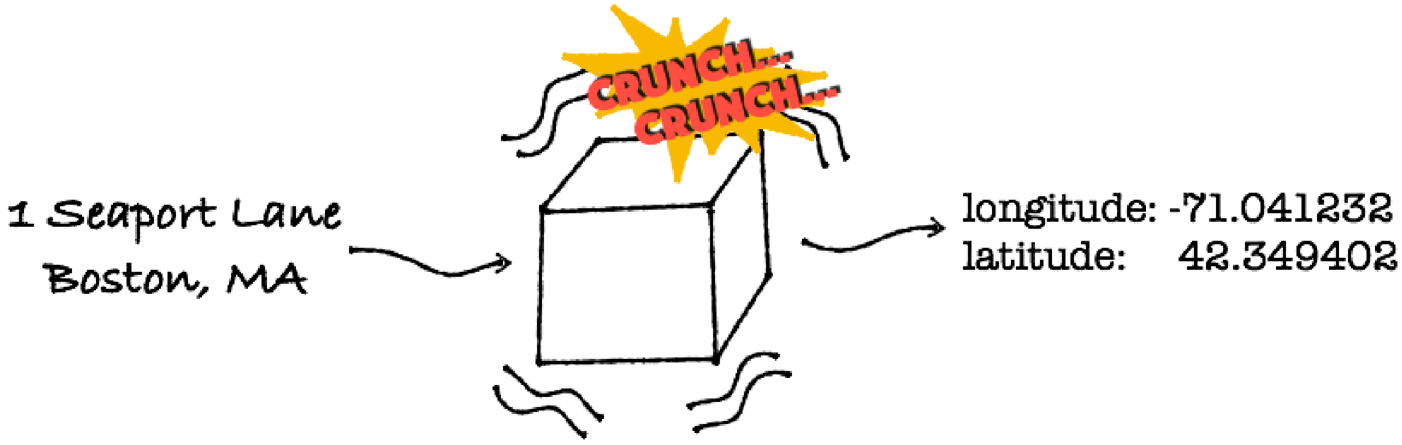
© 2017 Mapzen

Hosted Services are Hard (and So Can You!)

search (/tag/search) geocoding (/tag/geocoding) engineering (/tag/engineering)

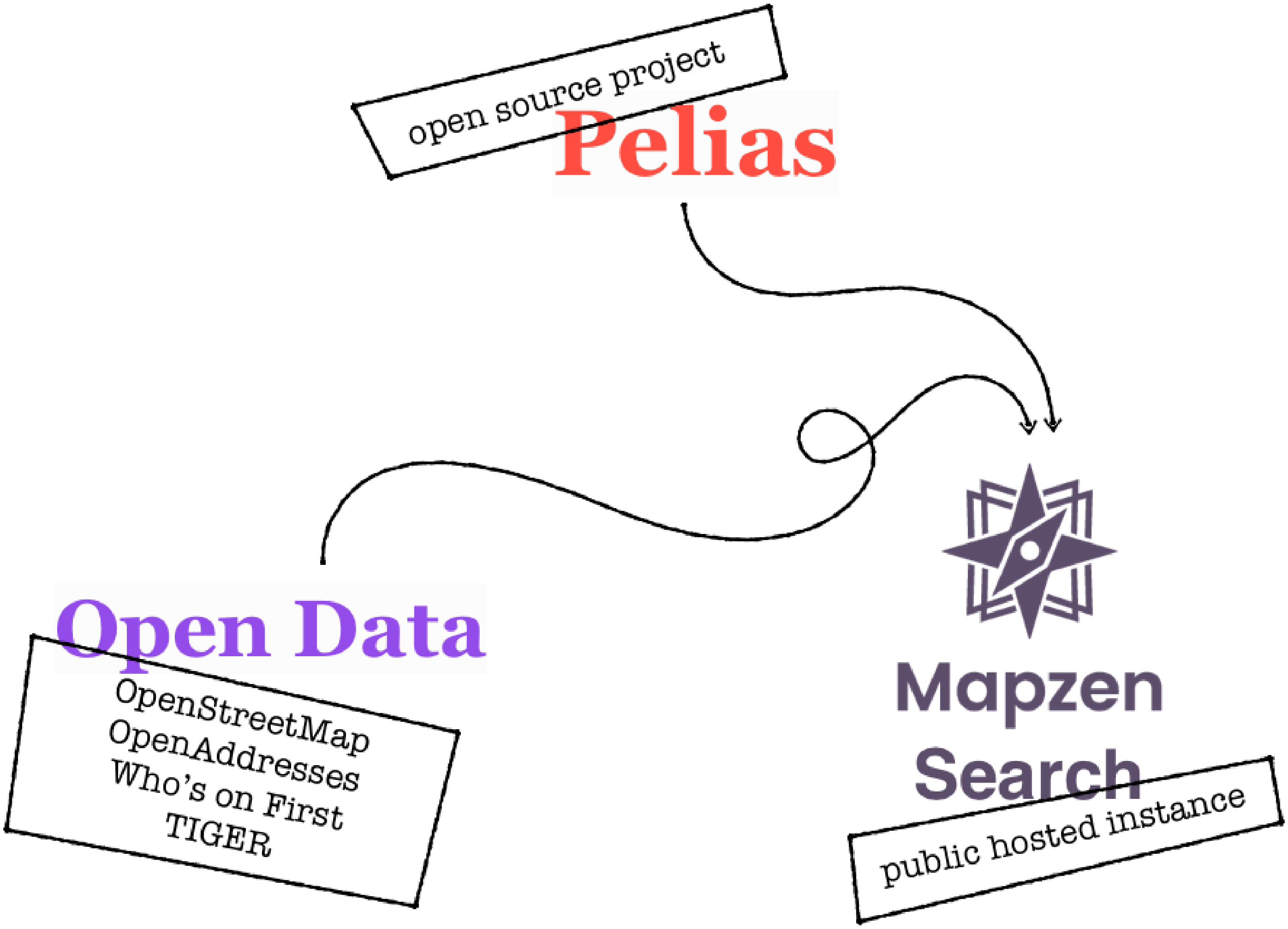
At the 2 year anniversary of the launch of Mapzen Search, the hosted instance of the open-source Pelias geocoder, we can't help but reflect on the journey. Building an open-source product, that uses live open data and is also a live service with live users, can be challenging. It often feels like there is a never-ending chain of big and small decisions that build on each other. We thought it might be informative to share some of the choices we're proud of and perhaps some that we've had to rework a few times. Get ready for a fast-paced tour through the evolution of an R&D open-source geocoder that became a commercial service, now handling millions of requests.

Let's start with a little background on what the Mapzen Search team has built. We work on a geospatial search engine known as a *geocoder*. A geocoder is a magic tool that can translate textual addresses into lat/lon pairs and vice-versa. It also understands venues and names of cities, regions, and countries.

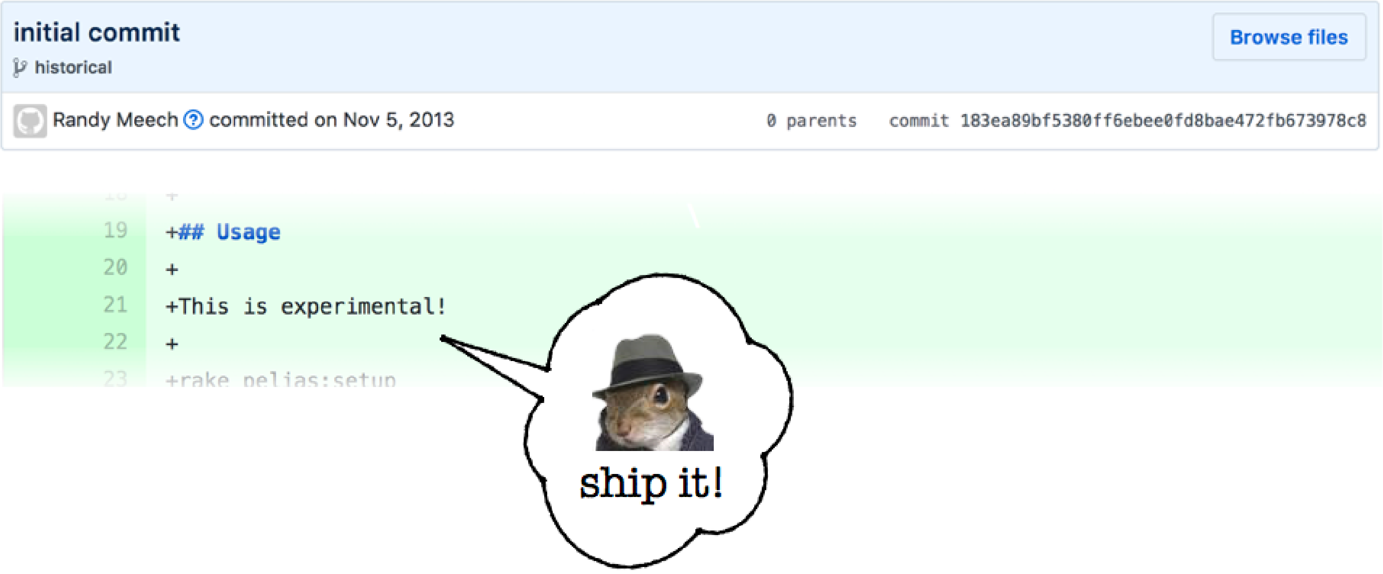


So we made one of these *geocoders*, and we are certainly not the first to do so. We named our geocoder Pelias... if you ever meet our founder Randy Meech, ask him why he did that 🤷‍♀️. We built our geocoder in the form of an HTTP service, so that applications anywhere connected to the internet could ask the geocoder about their addresses and places. HTTP services are a very accessible framework and we wanted to be **really** accessible.

Now, while the Pelias service can run on any network and use any data fed into it, in order to make it truly accessible we decided to host a public instance of it with only open data sources and allow anyone to access it from their web applications or other connected software. We call our hosted instance Mapzen Search and we ask our users to sign up for an API key in order to protect the service from getting overloaded with incoming requests by using rate limiting.


open source project
Pelias
Open Data
OpenStreetMap
OpenAddresses
Who's on First
TIGER
Mapzen
Search
public hosted instance

On November 5th, 2013 Randy Meech committed the first version of the Pelias geocoder to github. It looked very different at the time, written in Ruby, and had just a fraction of the functionality of today's engine. He also duly noted in the README that it was, in fact, experimental. It started as an R&D effort and we remain true to those roots today, with our excitement and willingness to innovate and continuously learn from our experimentation.



initial commit

historical

Randy Meech committed on Nov 5, 2013

0 parents commit 183ea89bf5380ff6ebee0fd8bae472fb673978c8

```
+
+## Usage
+
+This is experimental!
+
+rake pelias:setup
```

Later that year at the Code for America Summit, Randy unveiled the grand plan for Pelias and its core values.

> *Even in those early days, he had a vision for what a geocoder could be: open-source, built on open data, (in fact data-agnostic), and community driven!*

Two years worth of hard work later, at the Code for America Summit of 2015, Mapzen announced the release of the Mapzen Search API, which is a publicly-available hosted instance of the Pelias geocoder.

The hosted service was **officially launched on September 30th, 2015 (https://mapzen.com/blog/the-world-is-yours-announcing-mapzen-search/)**. We also tagged all the source-code with v1.0! At that point we felt that we reached a ceremonious end of our R&D journey. We accomplished the goal of a public geocoder, running on open data and available to all. Life was great! We even made beer mugs to celebrate.

1.0
PELI
FANCY

But the work was just beginning. Now there were real live users on the other side of those incoming requests! Now we had promises to keep. Now we had to represent the open data we were using in its true light. This was a whole new category of challenges for our team.

There are many things we learned along the way but some areas stand out more vividly as we look back. Each shows a trajectory and a desire to continuously improve our practices as well our products.

Documentation

We were particularly proud of our **Mapzen Search API documentation (mapzen.com/documentation/search)**. It was (and is) conversational, approachable, yet informative. We had done a ton of work on it, with the steady hand of Rhonda Glennon leading the way, before the 1.0 launch, since prior to that the only documentation we had was a markdown file that was accessible via one of the API endpoints. We had made a huge leap forward from those days. We even get compliments on it occasionally. Life was great!

> I’ve never seen an api with a methods param, this is amazing omg
>
> — Allen Tan (@tealtan) **April 24, 2017 (https://twitter.com/tealtan/status/856631802747899905)**

Then after a few months of work on some new functionality and a few minor releases, we came to an **embarrassing realization (https://mapzen.com/blog/release-notes/)** that we had not been keeping any release notes. Release notes are actually pretty important when working on an open source project! They let the world know what we’ve fixed or added in the software. For those using the open source project, it’s a tool for evaluating whether or not an update is necessary. For those using the hosted API, it’s a way to know why search results have changed and whether or not the client code needs to be updated in order to continue functioning correctly. So we quickly strapped those together for both the **Pelias project (http://pelias.io/release-notes.html)** and **Mapzen Search API (https://mapzen.com/documentation/search/release-notes/)**. Each requires a slightly different tone and level of detail. Pelias release notes need to include details about the changes made within the code, even when they are undetectable to the end-user. Mapzen Search release notes, however, have to denote updates to the builds and changes to the API protocol, such as new query parameters, for example. We made release notes a part of our workflow and life was good again!

Testing

If there is one thing our team wholeheartedly agrees on it’s *testing*! While the concept of testing software is not at all contentious, there are many ways to claim test coverage and our journey involved figuring out which strategies made us feel confident and covered.

About 2.5 years ago, long before the 1.0 release, the Pelias engine was covered with an impressive number of unit tests. Unit tests verify a single module or small slice of the codebase. These types of tests are allowed to be intimately familiar with the inner-workings of the code they are exercising. These tests are allowed to mock out all dependencies to ensure that only a single thing is being tested at a time. Unit tests are essential for complex parts of the codebase where it’s easy to make a mistake and assumptions about how the code should work need to be documented using nothing less than more code.

All of our unit tests are executed on a regular basis. We run them at every git commit, using pre-commit hooks. We run them using TravisCI anytime we push code to github. If any of the unit tests fail, the code changes cannot be accepted until all is fixed.

While unit tests are a great way to maintain individual parts of the code, they do very little to ensure that the overall system hasn't been compromised. They don't notify you when one module is not using another module correctly, or when the external data format has changed, thereby changing all the results. For that sort of verification, you need what's known as acceptance and functional tests. These types of tests treat the product as a black-box, not making any assumptions about the internals of the system and exercising it as an end-user would.

Acceptance tests are a special type of functional tests that an engineer writes when embarking on a new feature. They set the requirements for that new feature and the work can only be claimed complete once those tests are passing. Functional tests are all the acceptance tests the team has accumulated over the life of the product as well as any new ones that get added when a bug is found and fixed.

Looking back to that distant time 2.5 years ago, the search team wasn't using any functional testing strategies and was practicing what we came to call *feels-driven development*. Our functional testing strategy consisted of asking teammates to run their favorite queries manually and see if they *felt* the system was preforming better or worse. The only feelings this strategy ever resulted in were uncertainty and fear.

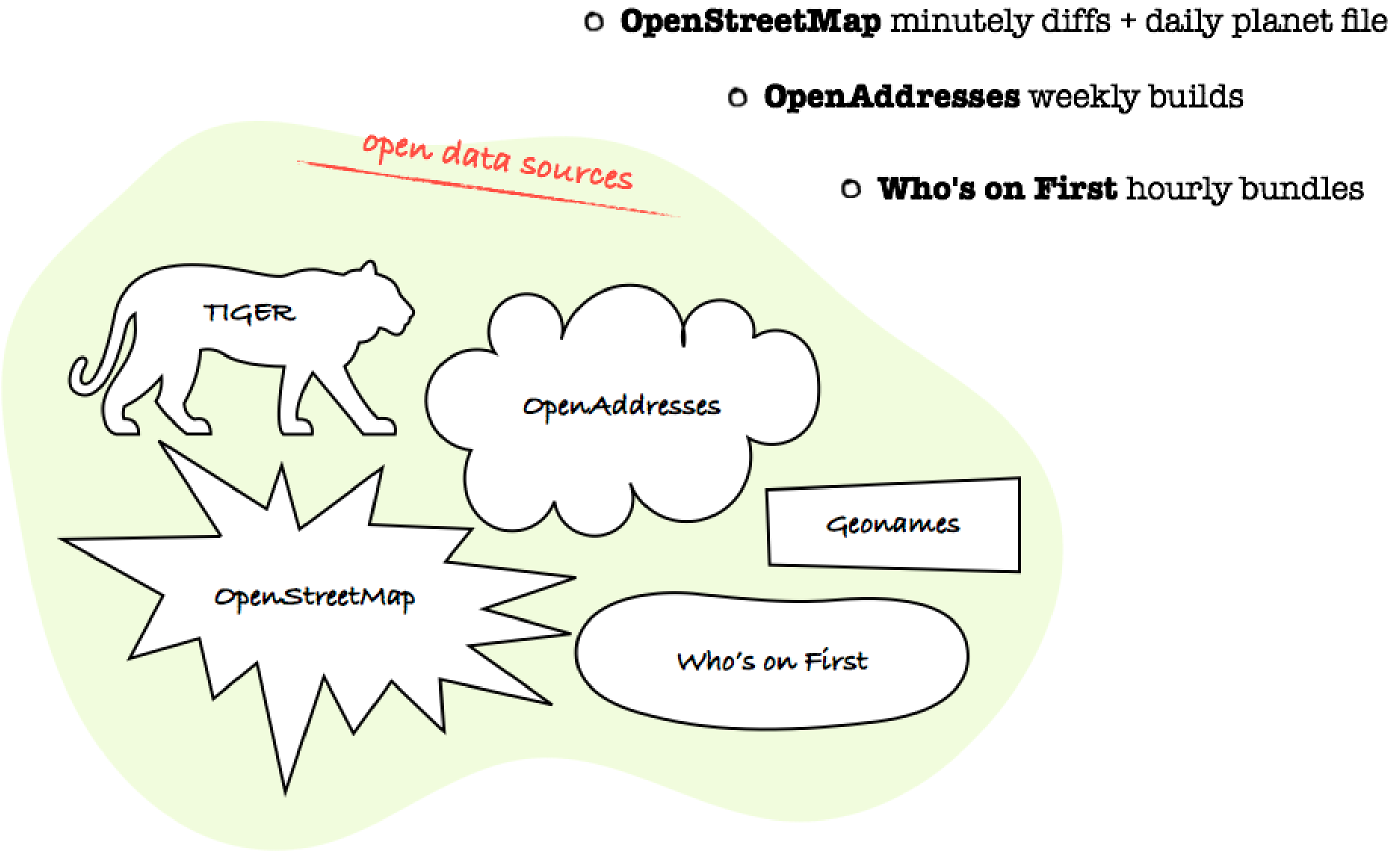
So we sat down and created a very simple framework for creating and running what would become our acceptance tests. This framework allowed us to specify input parameters, as users would when sending it queries, and then documenting an expected result. When the tests run, if the expected results don't match actual ones, the system yells at us about *regressions*.

We also have tests marked as known failures, which are placeholders for future bug fixes. If the recent changes made those known failures pass, the system congratulates us on *improvements*. This revolutionized the way the team worked and almost immediately instilled a sense of order and peace of mind. We knew we weren't going back on previously working functionality, and we could be confident that new features would be verified before ever making it to production. We really loooove <3 our acceptance test suite! It has helped us catch countless issues during development and staging, saving our users from having to experience them. We run these tests at every step along the development process and life is great, for the time being!

As we further improve our product and the work transitions from feature development to incremental quality and accuracy improvements, it will become increasingly difficult to capture the impact of many changes. This sort of trend analysis is much better done by machine than humans. We will need to rethink our current approach and supplement the existing strategies with new *data-driven* models. Data modeling, mining, and quantifying will be the building blocks with which we create this next generation of testing tools. *Data-driven development* will soon be the only reasonable way to account for a whole world of places. This approach will also help us identify gaps in quality using statistics instead of relying on manual bug reporting. #gloriousfuture

Data Imports and Build

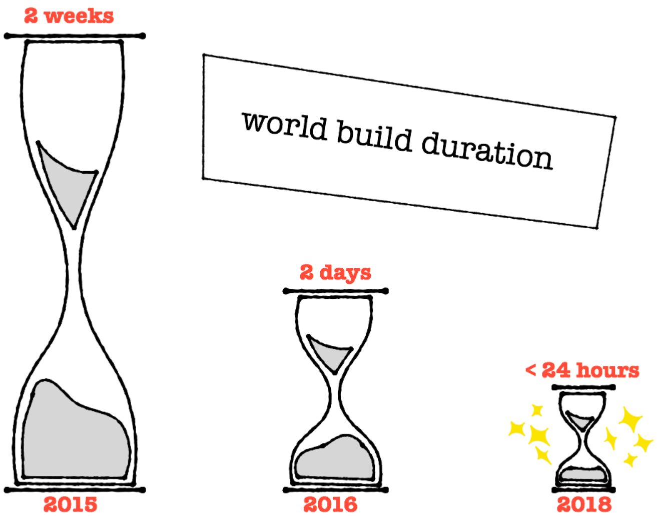
The hosted Mapzen Search API builds and serves only **open** geospatial data. At this time that data comes from four different and independent sources: **OpenStreetMap (https://openstreetmap.org)**, **OpenAddresses (https://openaddresses.io)**, **Who’s on First (https://mapzen.com/whosonfirst)**, **Geonames (https://geonames.org)**, and **TIGER (https://www.census.gov/geo/maps-data/data/tiger-line.html)**. Each of these sources is updated at different intervals and each publishes the updates in their own way. OpenStreetMap, for example, has minutely diffs and regularly updated planet files. OpenAddresses does a weekly build. Who’s on First has just updated their infrastructure to build the world bundles hourly.


OpenStreetMap minutely diffs + daily planet file
OpenAddresses weekly builds
Who's on First hourly bundles
open data sources
TIGER
OpenAddresses
Geonames
OpenStreetMap
Who's on First

When we talk about importing and building these various datasets into a single database, we're talking a **LOT** of data that needs preprocessing before it can be inserted into our backend data-stores. We are currently up to over 530M records just in our Elasticsearch index.

We do our best to represent all the data in its truest and freshest form, which currently means a weekly build across all datasets. Over 2.5 years ago, it was taking on the order of 2 weeks to build the data for the whole planet! That's a long time to wait to see if our code changes broke everything or fixed it. 🤞

Six months later we focused on optimization and brought that down to just under 30 hours, where we remain today! *Slow... clap...* But we're not done yet; we have set an internal goal to get to sub-24 hour builds by the end of this year, and we have every reason to believe that's entirely within reach.



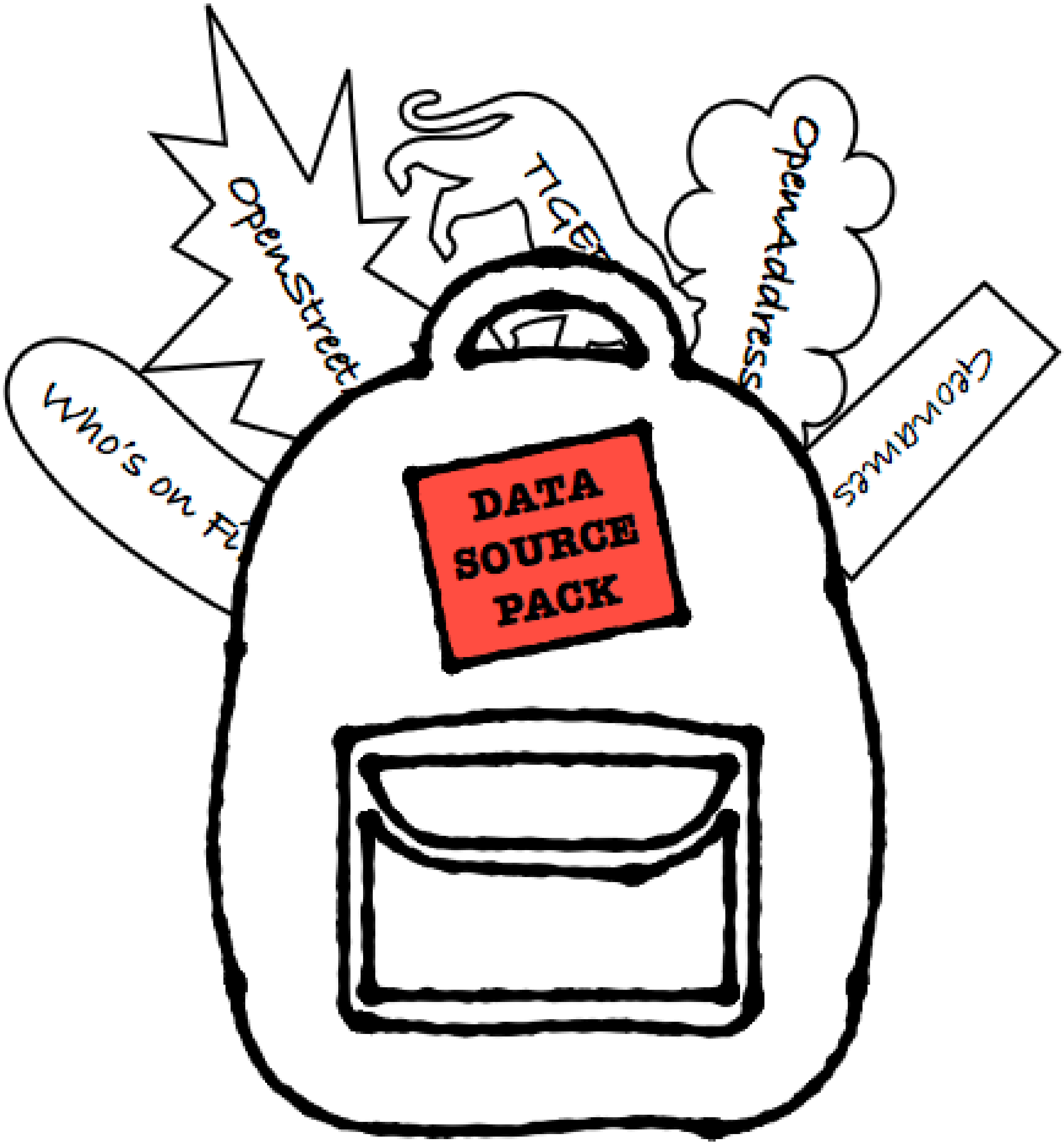
In the beginning of each build cycle, we download the most recent data directly from the source. We don't always control the contents or availability of the data, and download errors or corrupt bundles can be a big bummer for our build process.

For example, if a server is down and we cannot download OSM or Geonames data during our build process, the build cannot go on as planned. Manual intervention is required to figure out what went wrong and the whole process stalls.

Furthermore, if a single source's latest data download is unavailable for an extended period, the builds stop even if other sources are perfectly fine.

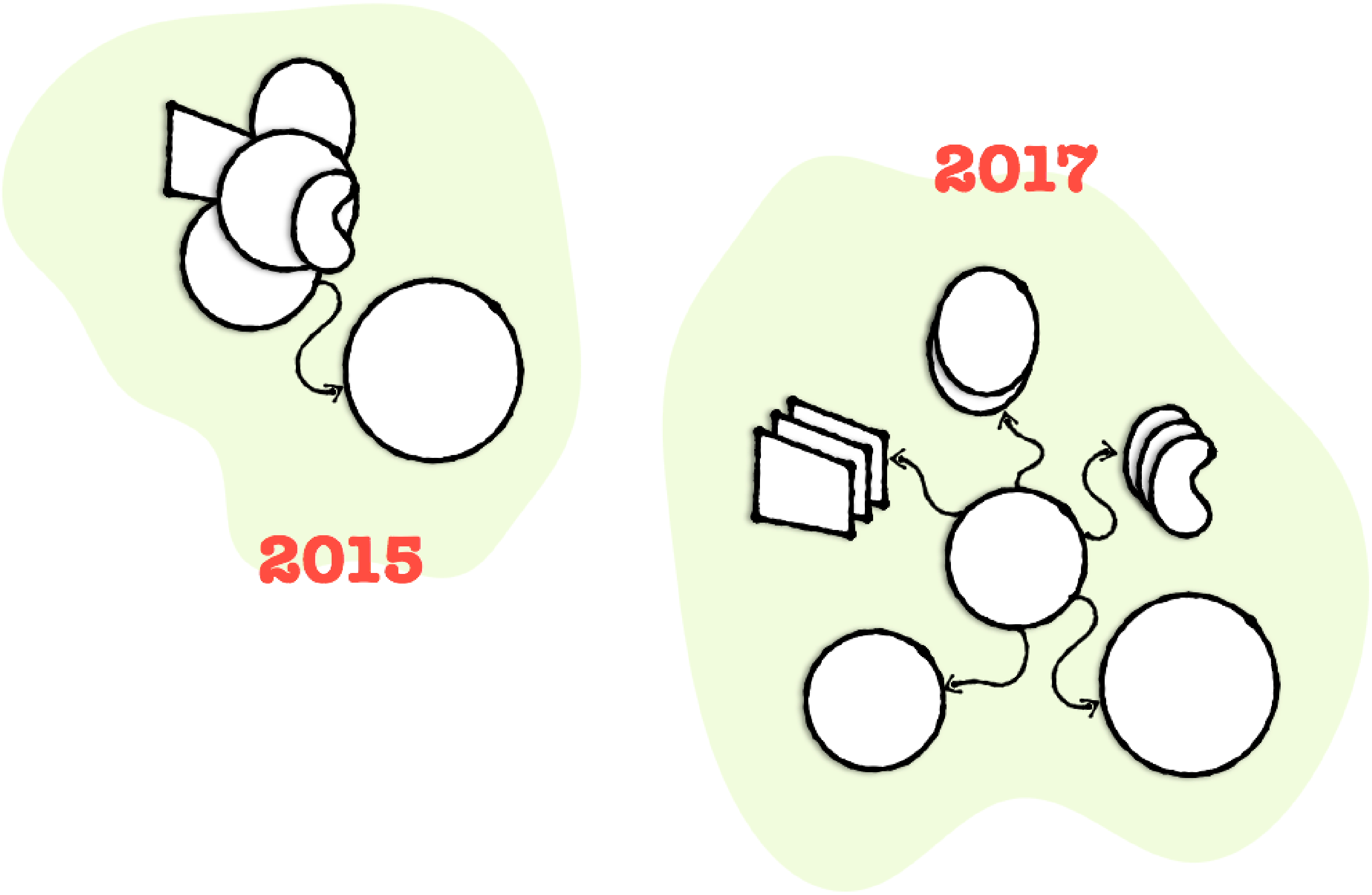
So we recently started a daily data download script that would collect and validate the data (as much as possible without a full build) from all sources. Builds then use the latest known good data when it comes time to build instead of downloading it directly from the source each time. This is an improvement, but we want to go further!

Our plan in the next year is to generate containers that have all the data inside them and can be mounted onto or referenced from any of our service machines, thereby creating what we've been referring to as *data-source-packs (the name will catch on eventually)*. The containers will be generated daily and we'll store some reasonable number of recent images in case the latest is causing trouble. This will also allow us to build different databases for each of our underlying parts of the engine using the same exact version of the source data. Without this important step, there is no guarantee that the data didn't change between the start times of each build process.



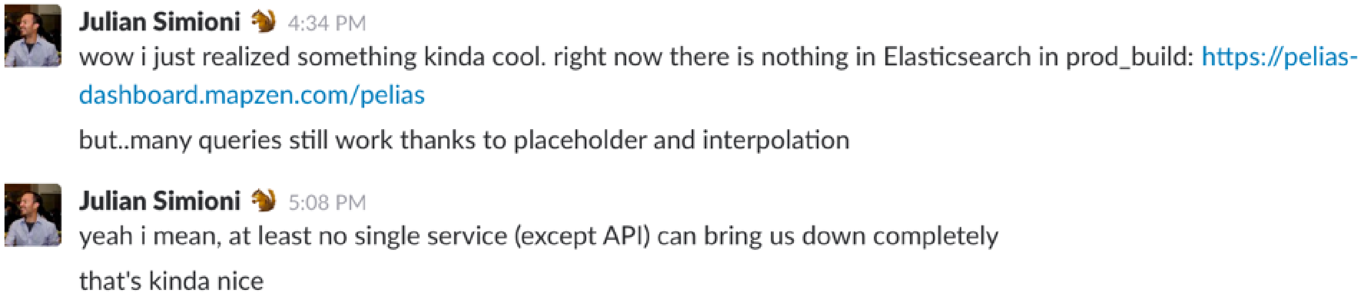
Microservices

Back in 2015, the Pelias API was a pretty classic monolithic codebase. All the functionality that the API required was implemented directly in the API module. The only external utility the API was accessing was the Elasticsearch datastore. Of course it was only a matter of time before the API became too large to maintain effectively. It also started requiring more memory per instance to support all the new and improved functionality. With the introduction of the libpostal address parsing utility the memory requirements became significantly higher per API machine. Then we brought all the admin geometries into the API in order to perform faster and more accurate point-in-polygon computations for our reverse geocoding functionality. That came with hefty memory requirements as well. It soon became obvious that we needed to systematically move those parts of the system into separate services.



With this distributed system come many benefits. Each service can be tested thoroughly without the blanket of the full API obscuring it. Each service can be deployed behind a load-balancer and only requires enough instances to support demand for that specific feature. In a monolithic system, in order to support more general requests we'd need to spin up more large machines that were idle a majority of the time. We also benefit from faster startup times for most services since they are only responsible for loading a small fraction of resources.

There was a recent revelation from one of our teammates that due to the way we had broken up the functionality of the API there was no longer a single point of failure and a good portion of the system now continues to work even when one of the major components is missing.



Julian Simioni 4:34 PM
wow i just realized something kinda cool. right now there is nothing in Elasticsearch in prod_build: https://pelias-dashboard.mapzen.com/pelias
but..many queries still work thanks to placeholder and interpolation

Julian Simioni 5:08 PM
yeah i mean, at least no single service (except API) can bring us down completely
that's kinda nice

Nowadays, when new features emerge they are usually developed as independent services and then integrated into the API using a lightweight service wrapper pattern. We love microservices!

While things aren't perfect and there are still many challenges to look forward to, life is pretty great! It's even more great when we consider how far we have come in just a couple of years time. We are proud of the work we are doing and hope to continue serving our users and the open-source community in a reliable and transparent way. Thank you for being part of our journey!

· 12 October 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

maîtres chez nous

whosonfirst (/tag/whosonfirst)

maîtres chez nous

aaron straup cope / nacis 2017

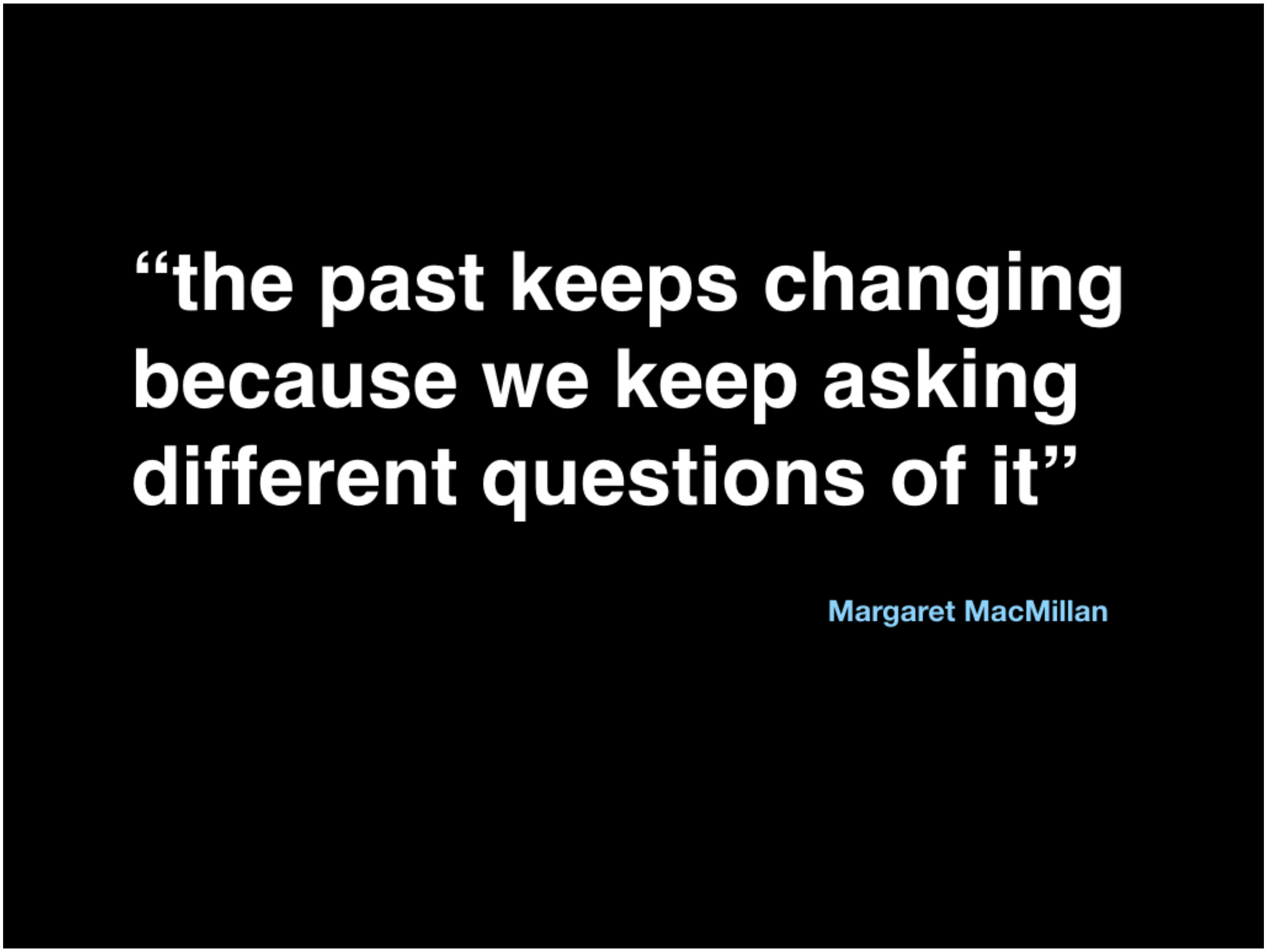
The following are the slides and notes for a talk about ***Who's On First (https://whosonfirst.mapzen.com)*** *that I delivered at the* ***2017 NACIS conference (http://nacis.org)*** *in Montreal.*

From the beginning our goal with Who's On First has not been to create a canonical list of carefully selected, important or relevant places. It is not meant to act as the threshold by which places are measured but rather as a welcoming vessel for all the places and the raw material by which any number of thresholds might be crafted. This was not a talk about how we do that work but rather why it is important to do it, at all.

Hello.

It is a pleasure to be back with you at **NACIS (http://nacis.org/)**. The last time I was here **was in 2011 (http://www.aaronland.info/weblog/2011/10/14/pixelspace/#nacis)**. It is also a treat for me that the conference is being held in **Montreal (https://places.mapzen.com/id/101736545/)** because this is where I grew up.

Thank you for inviting me to speak.



“the past keeps changing because we keep asking different questions of it”

Margaret MacMillan

This is a talk about gazetteers and in particular the gazetteer that I work on, called **Who’s On First (https://whosonfirst.mapzen.com)**. Who’s On First has set itself the ambitious goal of global coverage from continents all the way down to venues.

For anyone not familiar with the term a “gazetteer” is basically just a great big phone book, but a phone book of places rather than people.

Gazetteers have existed for as long as we've been able to conceptualize the idea of place. (http://www.iupress.indiana.edu/product_info.php?products_id=808056) A gazetteer is so deceptively simple as to seem mundane and beyond effort but so complex, the moment you scratch the surface, as to overwhelm even the best intentions.

This is not a talk about the mechanics of Who's On First. In the two and half years since the project began **something like forty to fifty thousand words have been written (https://whosonfirst.mapzen.com/blog)** about the architectural theory and the engineering choices and all the challenges we've faced along the way.

This is not a talk about how we work on a gazetteer like Who's On First but rather why we work on it, at all.

whosonfirst.mapzen.com

@allofthethings

@thisisaaronland

If not how but instead why, we still need to start with what. What is the shape of the Who's On First gazetteer?

- To begin, Who’s On First is an **openly licensed (https://whosonfirst.mapzen.com/docs/licenses/)** dataset. At its most restrictive, data is published under a Creative Commons By-Attribution license. Whenever possible, and this is true of our own day-to-day work, data is published under a Creative Commons Zero public domain license.
- Every record in Who’s On First has **a stable permanent and unique numeric identifier (https://whosonfirst.mapzen.com/data/principles/)**. There are no semantics encoded in the IDs.
- At rest, each record is stored as a plain-text GeoJSON file. Our goal is to ensure that Who’s On First embodies the principals of portability, durability and longevity. This led us to adopt plain-vanilla text files as the base unit of delivery.
- Files are stored in a nested hierarchy of directories derived from their IDs.
- There are **a common set of properties (https://whosonfirst.mapzen.com/docs/properties/)** applied to all records which may be supplemented by an arbitrary number of additional properties specific to that place.
- There are **a finite number of place types (https://github.com/whosonfirst/whosonfirst-placetypes#here-is-a-pretty-picture)** in Who’s On First and all records share a common set of ancestors. As with properties, any given record may have **as complex a hierarchy as the circumstances demand (https://whosonfirst.mapzen.com/docs/hierarchies/)** but there is a shared baseline hierarchy across the entire dataset.
- Individual records may have multiple geometries or multiple hierarchies and sometimes both.
- Records may be updated or superseded, cessated or even deprecated. Once a record is created though it can never be removed or replaced.
- Lastly and most importantly Who’s On First is meant, by design, to accommodate all of the places.

Who’s On First is not a carefully selected list of important or relevant places. It is not meant to act as the threshold by which places are measured.

Who’s On First, instead, is meant to provide the raw material with which a multiplicity of thresholds might be created.

From Who's On First's perspective **the point is not that one place is more important or relevant than any other (https://mapzen.com/blog/who-s-on-first/)**. The point is not that a place may or may not exist anymore or that its legitimacy as a place is disputed. The point is, critically, that people *believe* them to exist or to have existed.

This is why I often refer to Who's On First as **"a gazetteer of consensual hallucinations" (https://mapzen.com/blog/mapping-with-bias/)**.

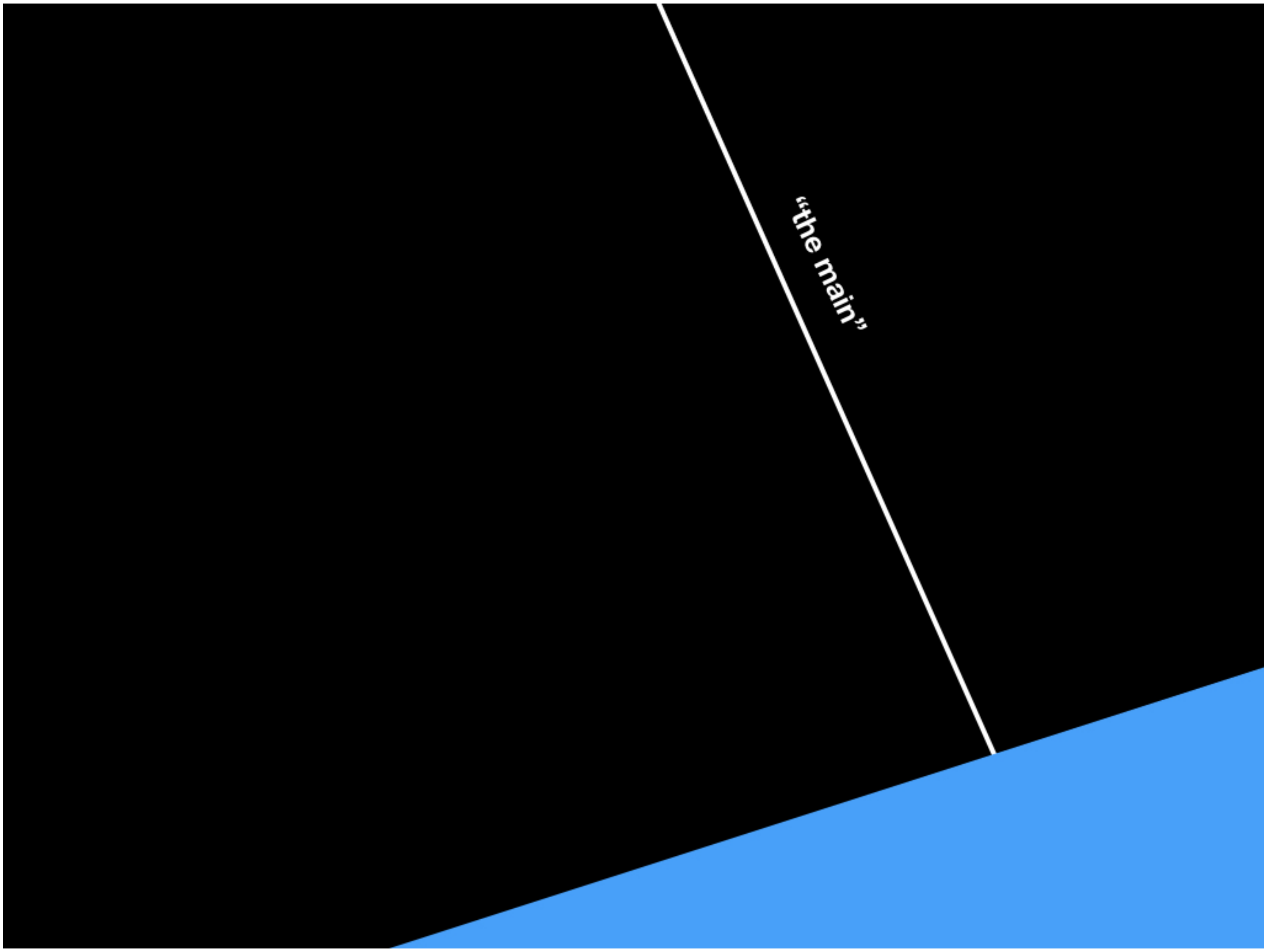
PAGES
Kevin Ledo

To explain why I'm going to start with this guy.

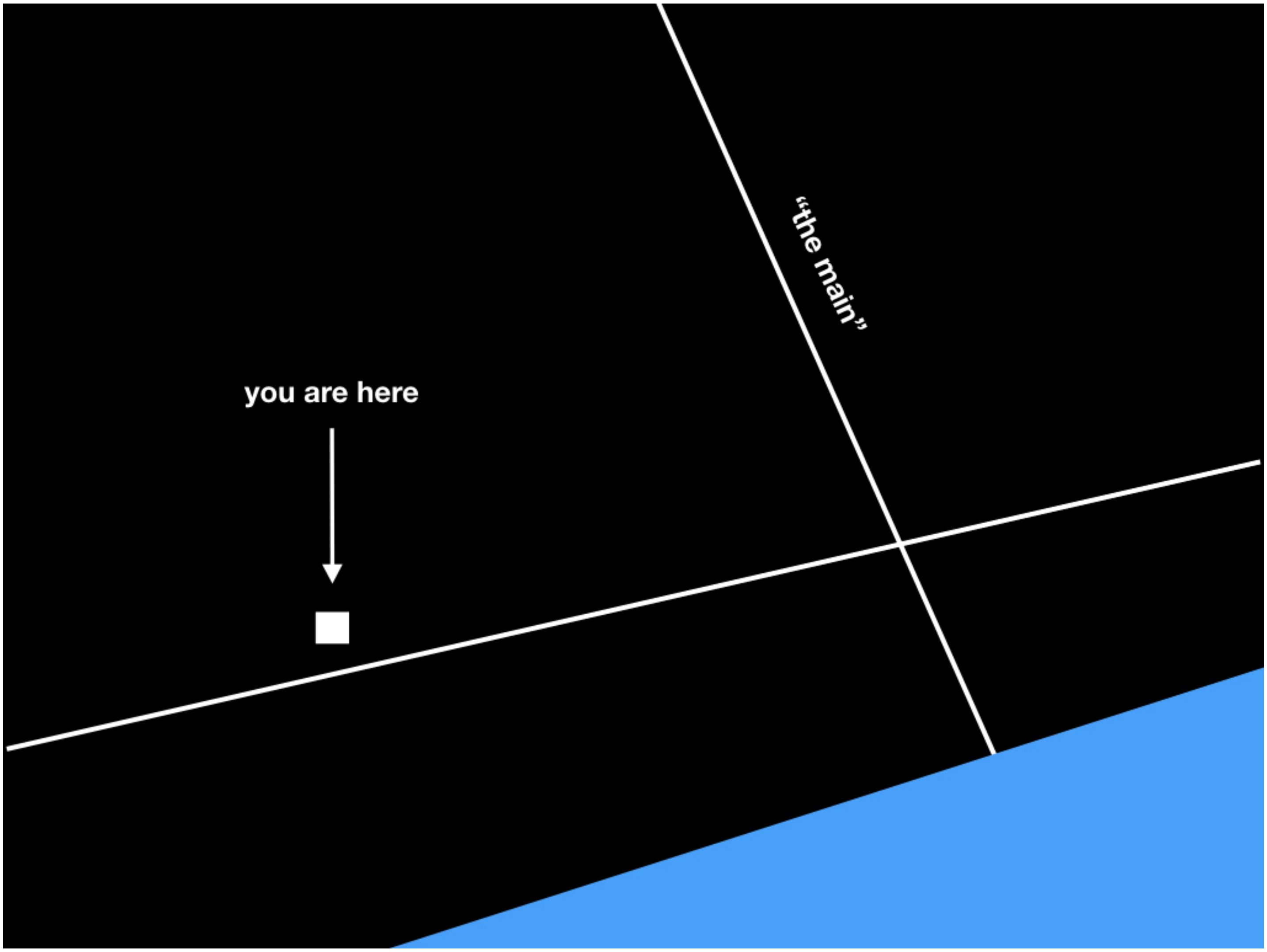
This is **Leonard Cohen (http://www.pbs.org/video/montreal-ladies-and-gentlemen-mr-leonard-cohen/)**, the poet and musician, who **died in November (https://www.youtube.com/watch?v=jP7pN_3_JQI)** of last year. Cohen was born and raised in Montreal and **this mural (http://montrealgazette.com/news/local-news/leonard-cohen-looms-large-at-montreals-mural-international-public-art-festival)** was **painted**

(http://www.kevinledo.com/) on St. Laurent Boulevard, just **down the street (https://www.nytimes.com/2017/02/10/travel/leonard-cohen-musician-montreal-canada.html)** from where he kept a house in the city.

St. Laurent Boulevard is often referred to by locals simply as “The Main”.



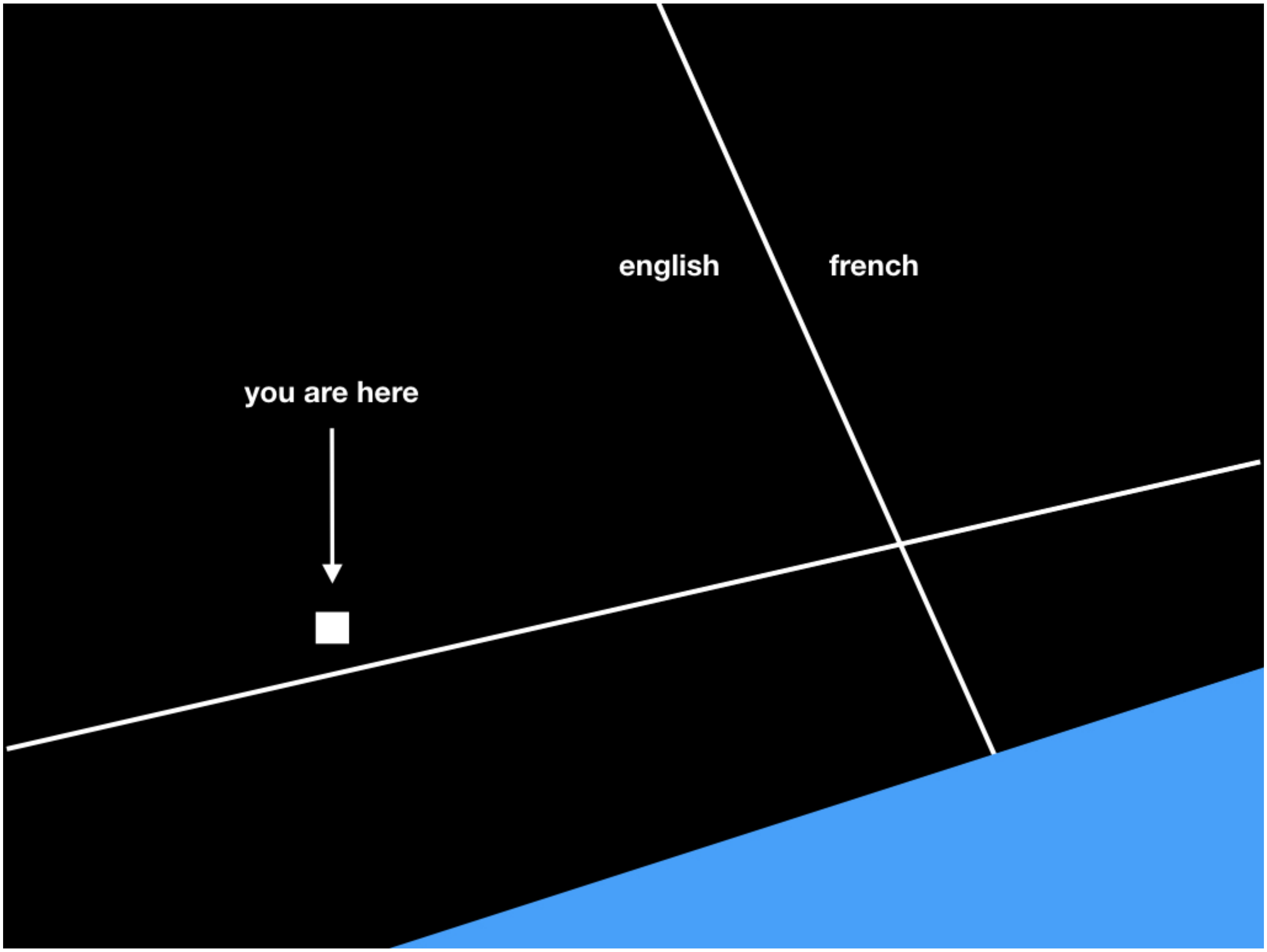
The Main is so-called because it is one the city’s oldest streets running along a North-South axis, starting in Old Montreal at the bank of the **St. Lawrence River (https://places.mapzen.com/id/404529181/)**.



Like a lot of cities in the world, Montreal's North-South axis has no bearing in reality.

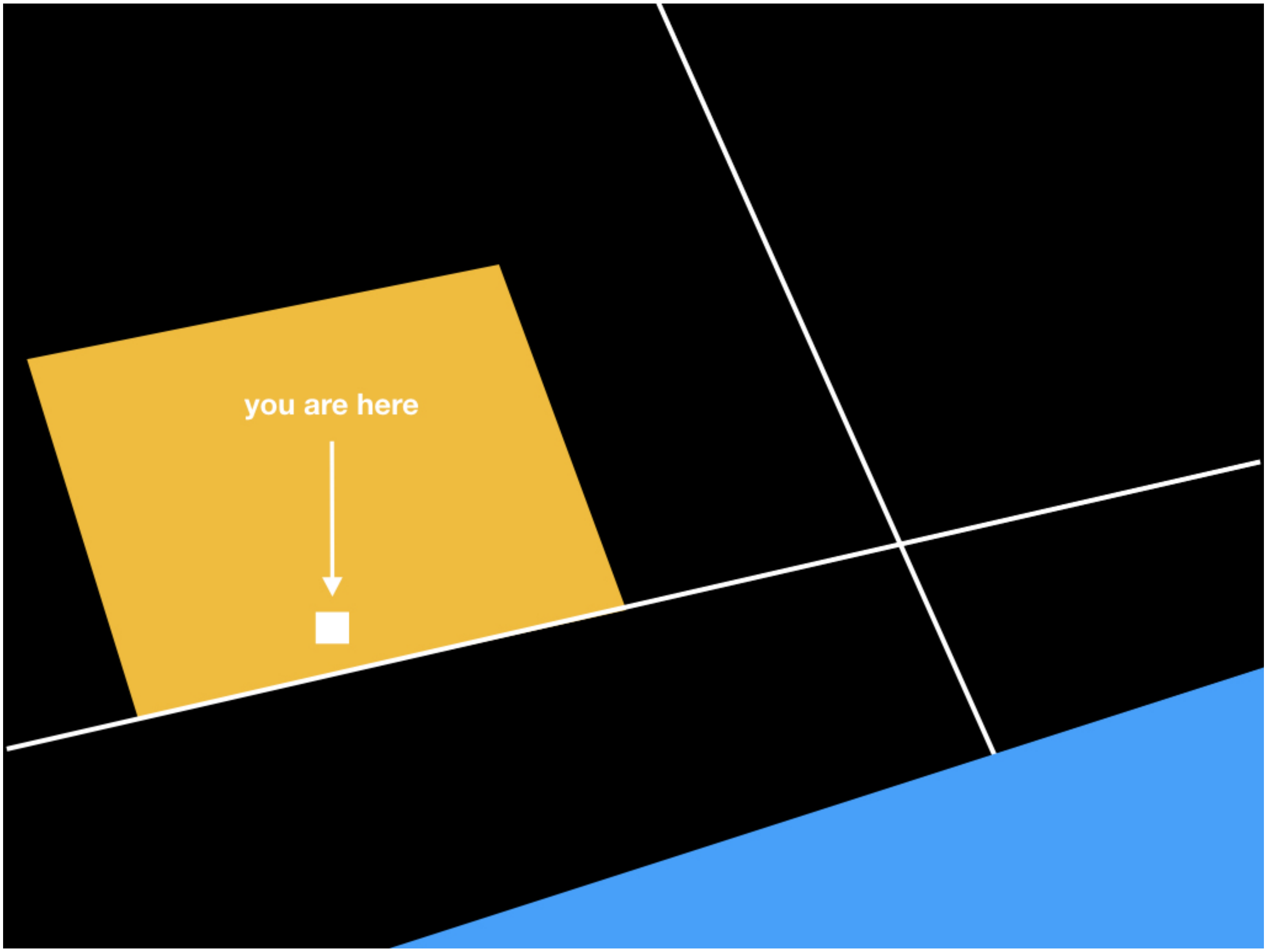
If you're being precise St. Laurent Boulevard runs along a North-North-West to South-South-East axis.

What's important about The Main is less its directionality than the fact that it acts as the border between the Eastern and Western halves of the city.



It is only in recent years that The Main has not served as the dividing line between the city's English and French speaking populations.

This was famously described by the author **Hugh MacLennan (http://www.hipmuseum.com/hugh.html)** in the larger Canadian context as "two solitudes" living side by side **with little or no common ground between them (http://archive.gg.ca/media/doc.asp?lang=e&DocID=4574)**.



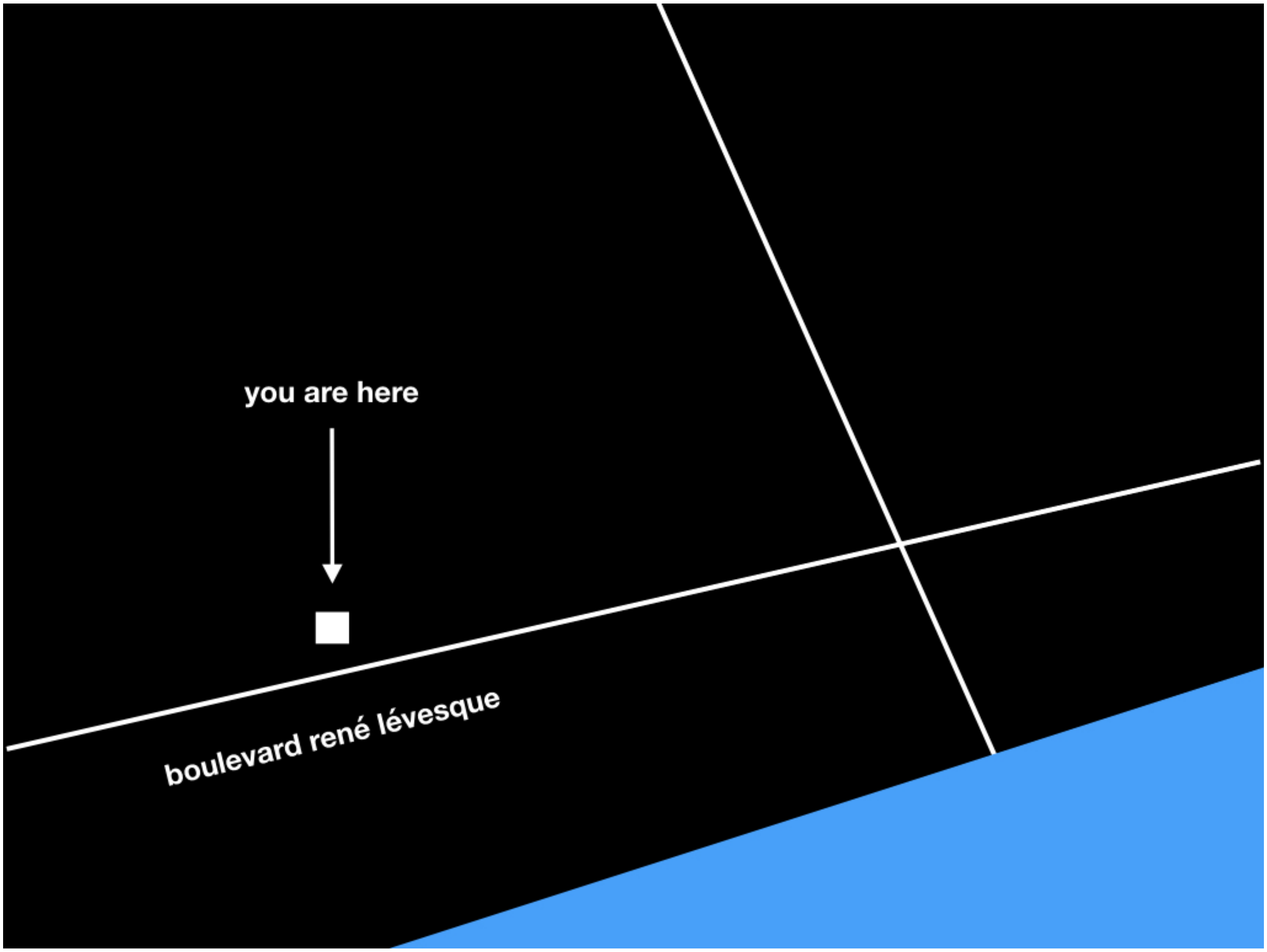
It is also not really possible to talk about that divide without talking about it as the border between the haves and the have-nots.

This conference is being held on the Southern edge of what was once called **The Golden Square Mile (https://places.mapzen.com/id/1108955947/)**. This part of town is where what we now call "the 1%" lived.

For a time this was the seat of all the wealth and power in Canada. It was very very English at a time when the contempt and disdain between the English and the French was barely contained and rarely disguised.

This created a power dynamic that rippled down through every aspect of society, in a province that has always been 90% Francophone. It was a power dynamic that was not only allowed to **fester (https://www.urbandictionary.com/define.php?term=T%C3%AAte%20carr%C3%A9e)** but actively promoted.

This continued to varying degrees until the 1950s and what is known as **the Quiet Revolution (http://archives.radio-canada.ca/politique/provincial_territorial/dossiers/1732/)** when the French Canadian population demanded, not for the first time, to be **“maîtres chez nous” (https://en.wikipedia.org/wiki/Quiet_Revolution)** or “masters in our own home.”



You may have noticed that the conference hotel sits on a street named after a guy called **René Lévesque (https://en.wikipedia.org/wiki/Ren%C3%A9_L%C3%A9vesque)**.

It is difficult to overstate Lévesque’s role during **the Quiet Revolution (http://www.banq.qc.ca/collections/collections_patrimoniales/bibliographies/revolution_tranquille.html)** and in shaping contemporary life and the social contract in Québec.

He would be elected Premier in 1976 as the head of the separatist Parti Québécois (PQ) and would lead **an unsuccessful bid to secede from Canada in 1980 (https://www.youtube.com/watch?v=lru4grpq3Rc)**.

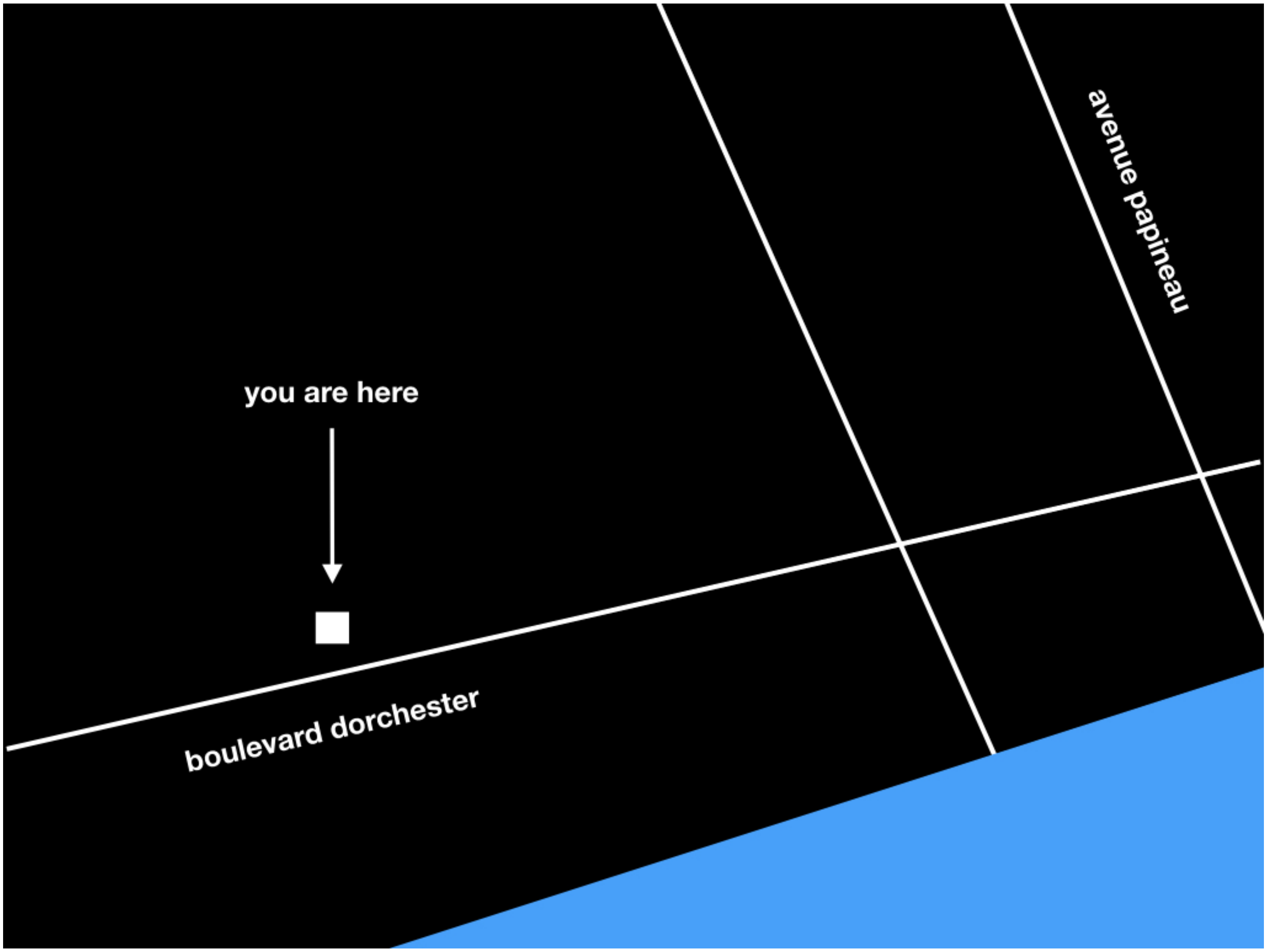
Note: During the talk I incorrectly stated that "Among other things he gave Québec a Charter of Human Rights and Freedoms almost a decade before Canada ***enacted its own (http://laws-lois.justice.gc.ca/eng/Const/page-15.html)*** *and language laws which are still being debated, to this day."*

In fact, The ***Québec Charter of Human Rights and Freedoms (https://en.wikipedia.org/wiki/Quebec_Charter_of_Human_Rights_and_Freedoms)*** *was passed by the provincial National Assembly the year before the PQ were elected. That is my bad so I will point out, instead, that Lévesque both championed and oversaw the final* ***nationalization of hydroelectric power in Québec (http://larevolutiontranquille.ca/en/the-nationalization-of-electricity.php)****. While not necessarily as inspiring as a Charter of Human Rights and Freedoms it would have an equal (probably greater) impact on the province.*

By the time he died in 1987 few would deny Lévesque, whatever they thought of his politics, his place as **the father of modern Québec (https://www.youtube.com/watch?v=BhbSp4cJZVY)**.

So you would be forgiven for wondering why we have chosen to honour such a celebrated figure with such an unremarkable boulevard. The next time you go outside take a moment to marvel at its mediocrity.

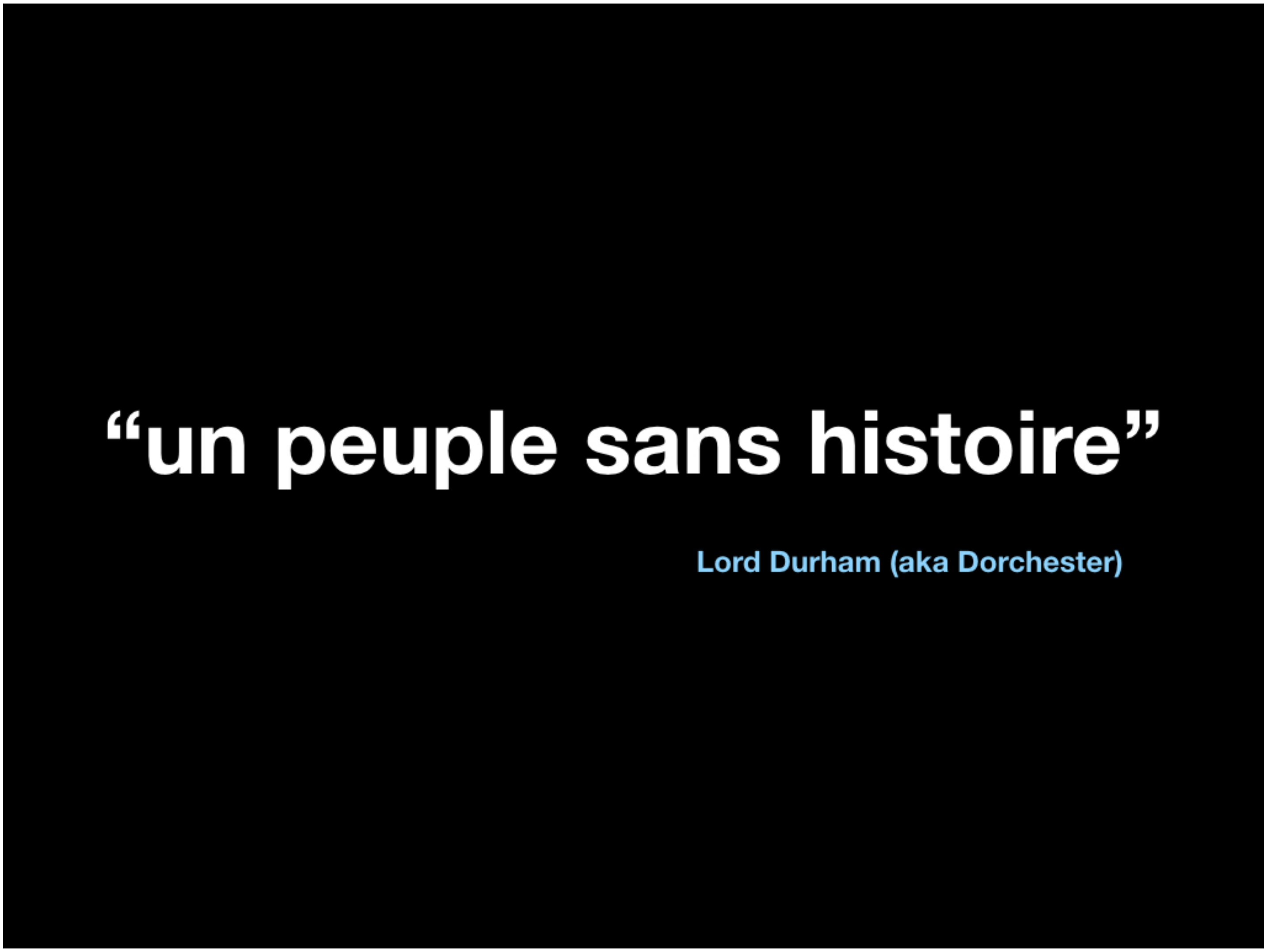
The answer lies in what Boulevard René Lévesque used to be called.



Boulevard René Lévesque used to be called Dorchester Boulevard after the English Lord Dorchester, who was formerly known as Lord Durham.

Lord Durham was sent over to Canada in 1838 to report on **the Patriot Rebellions (https://fr.wikipedia.org/wiki/R%C3%A9bellion_des_Patriotes)** of the previous year and more generally to determine why the Crown's French Canadian subjects were getting restless.

Fun fact: The rebellions were led by **Louis-Joseph Papineau (https://web.archive.org/web/20170612175735/https://www.pc.gc.ca/en/lhn-nhs/qc/louisjosephpapineau)** who was also honoured with an equally drab and uninspiring avenue about 40 minutes walk from here.



“un peuple sans histoire”

Lord Durham (aka Dorchester)

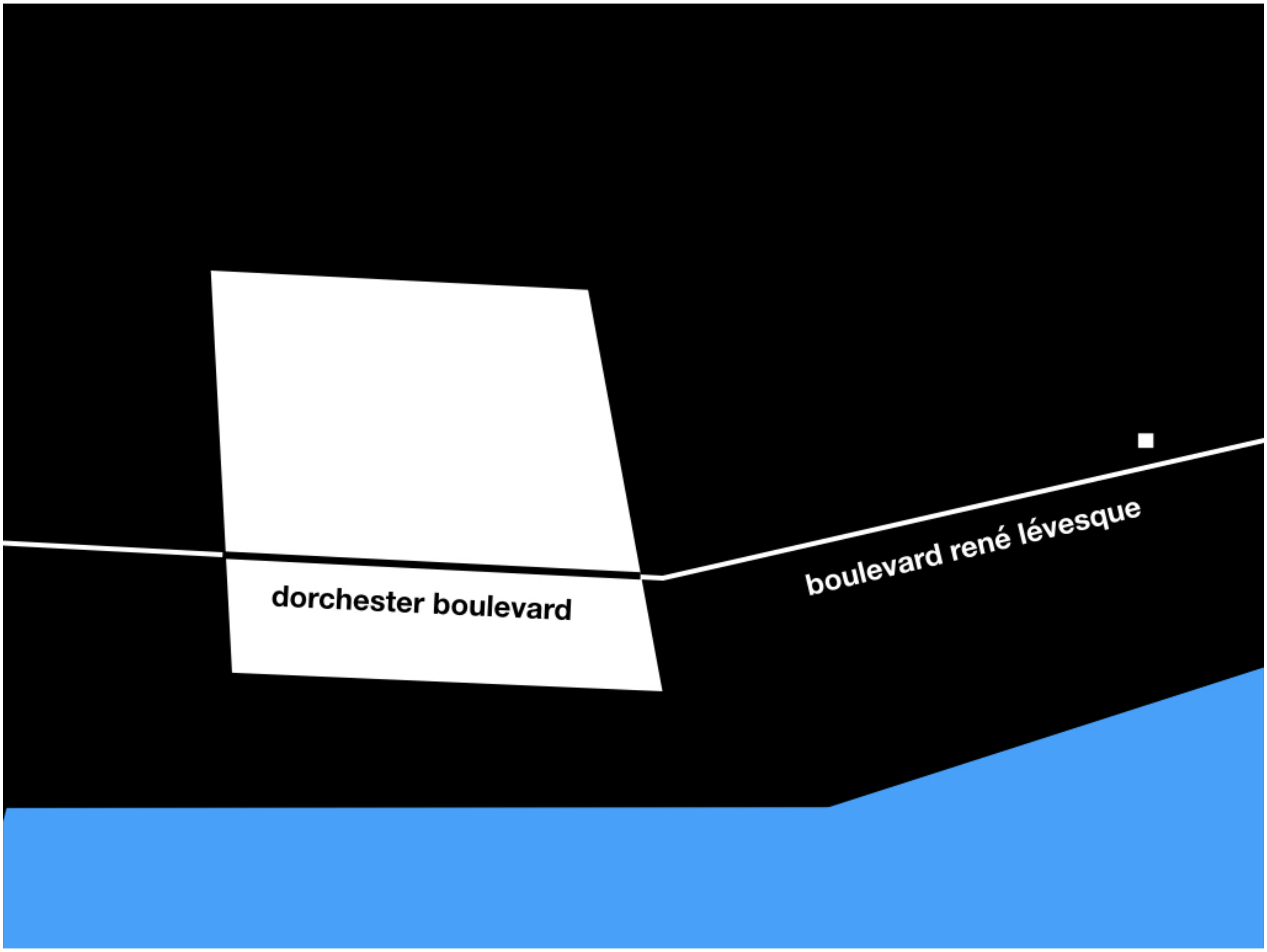
Lord Durham’s report (https://en.wikipedia.org/wiki/Report_on_the_Affairs_of_British_North_America) might have faded into historical obscurity had its author not offered the opinion that the French Canadians, who predated the first English settlers by a hundred years or more, were “un peuple sans histoire” or “a people without history.”

“je me souviens”

everyone else

It is not for nothing that all the license plates in this province now read **“je me souviens” (https://www.youtube.com/watch?v=2FZXQZJxZRk)** or “I remember”.

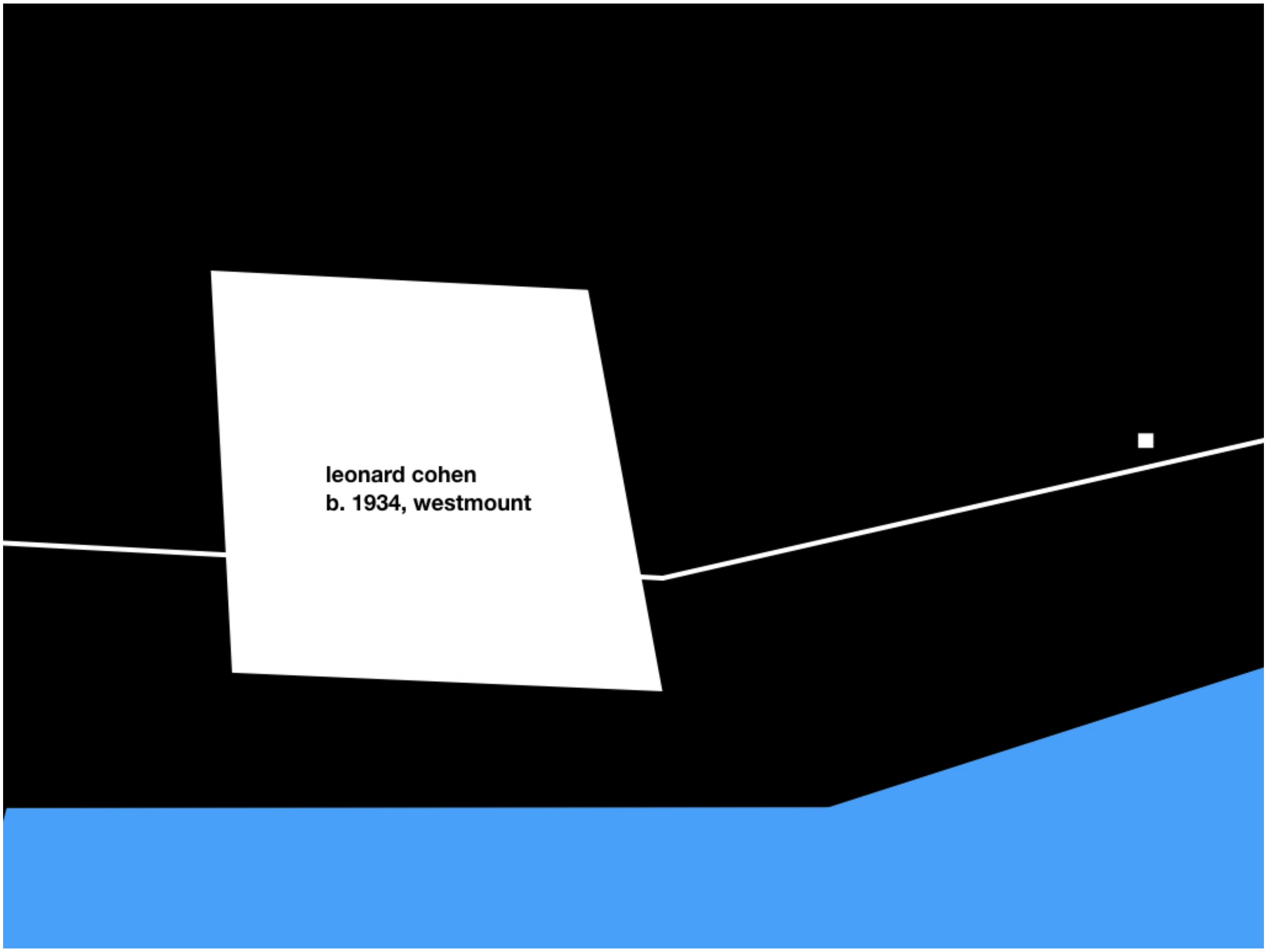
Nor is it surprising that Dorchester Boulevard was purposely renamed to honour René Lévesque.



You know who else remembers, though? The city of **Westmount (https://places.mapzen.com/id/101736451/)** remembers.

Westmount is **an autonomous municipality (https://westmount.org/en/the-city/history/)** on the island of Montreal and home to the remaining money and power that didn't immediately decamp to **Toronto (https://places.mapzen.com/id/101735835)** following the **election of René Lévesque (http://collections.musee-mccord.qc.ca/en/collection/artifacts/P090-A_50-1004/)**.

Westmount is a sort of leftover bronze square mile of wealth and privilege and in 1987 it **refused to rename Dorchester Boulevard (https://www.westmount.org/wp-content/publications/A_View_Of_Their_Own_the_Story_of_Westmount/files/assets/basic-html/page151.html)** as the rest of the city had, and so it remains today.

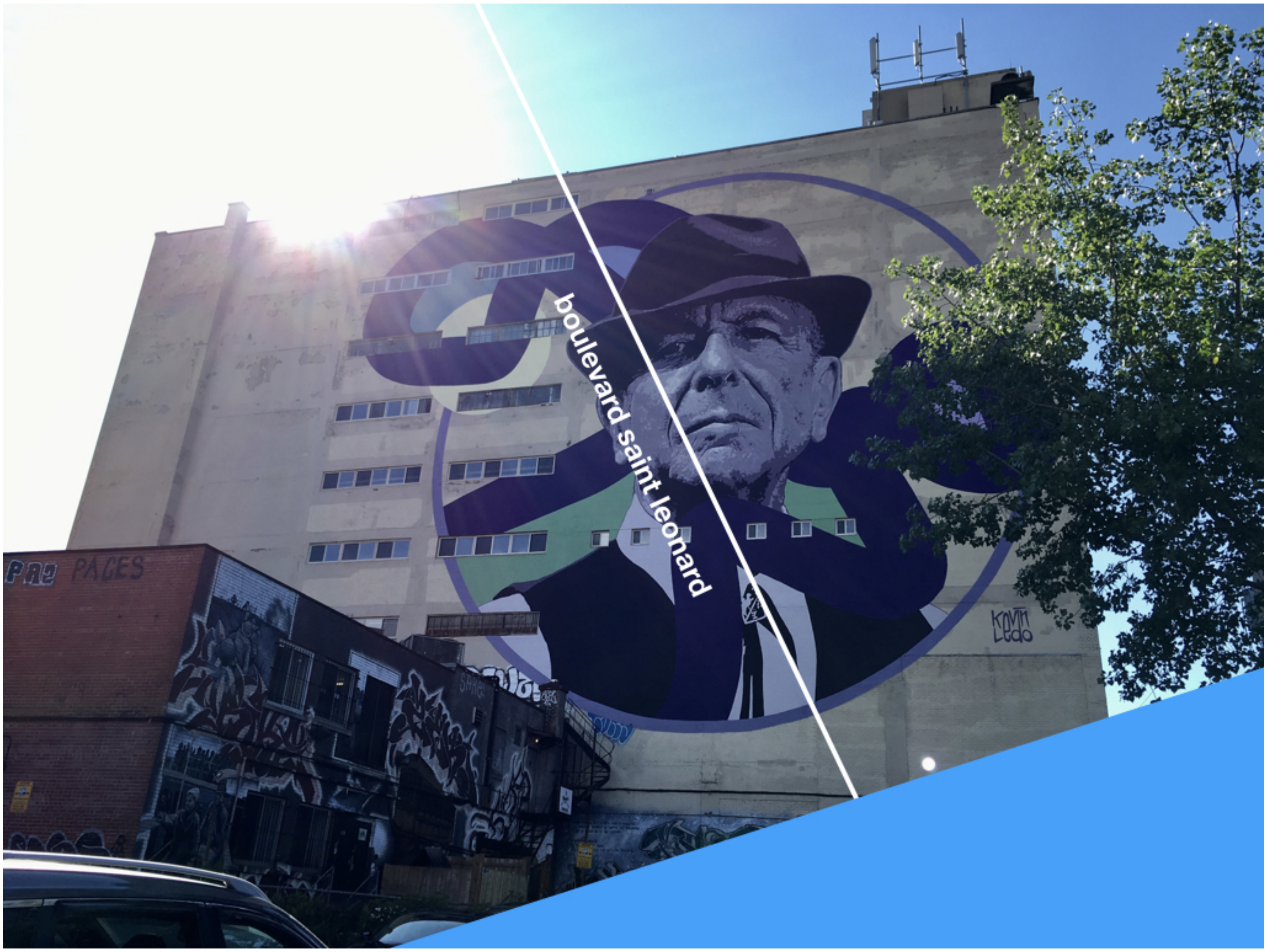

leonard cohen
b. 1934, westmount

Fun fact: **Leonard Cohen was born in Westmount (http://www.leonardcohenfiles.com/montreal.html)**, in 1934.

So, we’re back to this guy.

By the time Leonard Cohen died last year in Los Angeles **he had lived there for decades (http://www.cbc.ca/news/entertainment/leonard-cohen-la-home-1.3848347)**. This pesky fact has never prevented anyone in Montreal, English or French, from claiming him **as one of their own (https://www.newyorker.com/culture/culture-desk/leonard-cohens-montreal)**.

In the minds of many people in this city Leonard Cohen is before all things **a Montrealer (https://www.youtube.com/watch?v=ibUKEwds4Ng)**.



It is in all seriousness then that I would like to propose that the city of Montreal rename St. Laurent Boulevard after Leonard Cohen.

The simple fact is that Montreal suffers from a surplus of streets and plazas and landmarks devoted to the religious and linguistic power struggles and score-settling of the past.

We are overdue something that reflects the present. **We are overdue a celebration (https://www.youtube.com/watch?v=2FpwjQLZTTs)**.

I have told you these stories for two reasons:

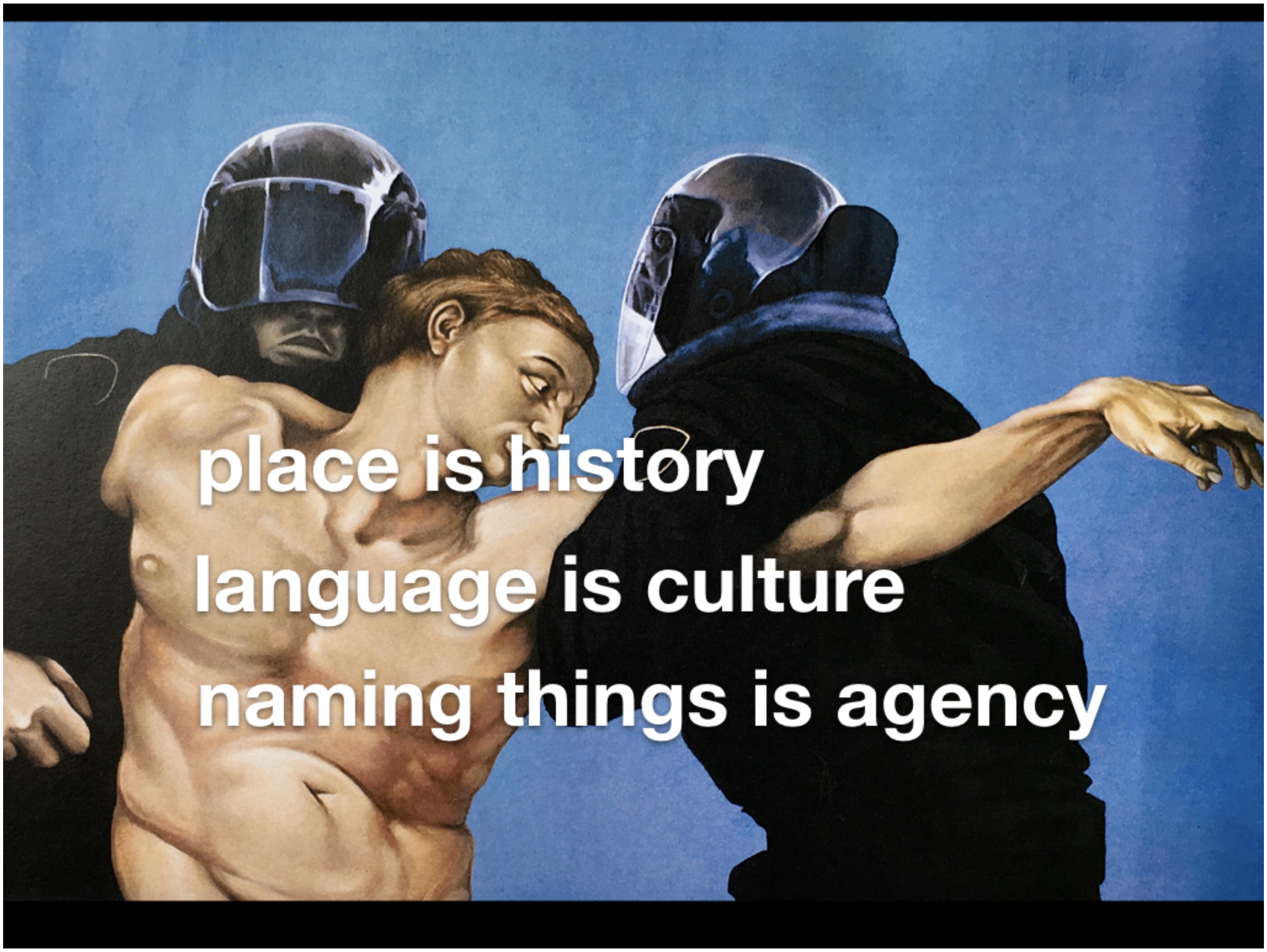
The first is that all of these things really happened and they are the stories that inform the city you'll be visiting during this conference.

The second is to use the telling – and in particular the telling of a duality, of ”two solitudes" and “two founding nations” – to highlight an uncomfortable fact about Canadian history, French and English both.

Simply put, we have been blind to the fact that there has always been another story. We have been blind to the fact that the First Nations were already here living on these lands long before the European settlers arrived.

It is important to recognize that we have not been passive in our blindness but brutally deliberate. First out of malice and then later out of negligence and more recently out of shame.

We have closed our eyes to the fact that almost from the beginning we have systematically worked to erase the First Nations, hoping only to open them when the deed is done and there is nothing left to remember but the two solitudes.



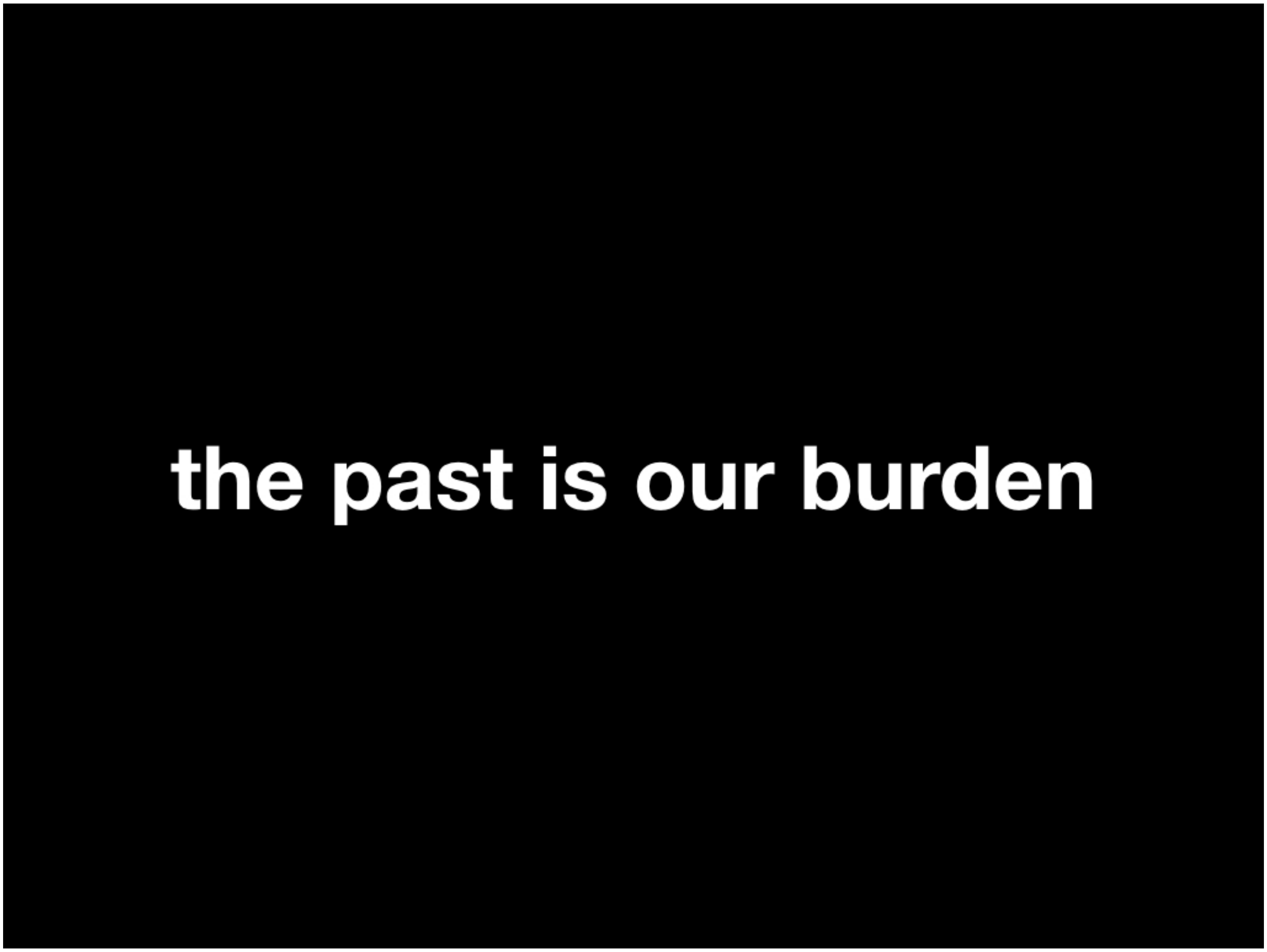
In 2017 there is at least a nascent awareness, something approaching acknowledgement, among Canadians that **these things happened and that we did them (https://gadebate.un.org/en/72/canada)**.

We are a long way from making amends and **there is not time enough to catalog all the ways we have done wrong (http://www.trc.ca/websites/trcinstitution/index.php?p=3)** by the First Nations but I want to touch on one story in particular.

As late as the 1950s and the 1960s it was, as part of official government policy to assimilate the First Nations, our practice **to forcibly remove First Nations children from their families (https://www.nytimes.com/2017/10/06/world/canada/indigenous-forced-adoption-sixties-scoop.html)**.

These children were placed in **foster homes or residential schools (http://www.aadnc-aandc.gc.ca/eng/1100100015576/1100100015577)** where in addition to widespread physical and sexual abuse they were actively prevented from learning or practicing or remembering their language, their culture, their histories and in many cases their own birth names.

If you believe that place is history or that language is culture or that naming things is an important form of agency – *all things which are the very stuff that maps are made of* – it is difficult not to see those policies as anything but a sanctioned and pre-meditated erasure.



the past is our burden

I take the view that the sins of a mother or a father are not those of their sons or their daughters. We have enough evidence to suggest that things never end well when we assume they are. I do not believe however that this allows those same sons and daughters to abdicate their past.

The past I am sorry to say is our burden.

It is our burden because while the stories I've told you today have taken place in a comparatively modest slice of geography to a comparatively minuscule population *they are the same story everywhere you go*.

The details may change but the story is always the same.

If there is one common thread that binds us all it is that we argue about what to call a place and the meaning of those names.

If there is another common thread it has been to assume the argument is a zero-sum game.

better than yesterday
LOVE YOU WHEN YOU'RE UGLY

What if we stopped doing that?

I have described to you a series of overlapping grievances and past injustices that *in no way* will a gazetteer remedy alone. A gazetteer is only a tool but perhaps we can stop teaching our tools the bad habits of the past.

A gazetteer is fundamentally about the relationships between one place and all the others. In the past we have been limited by economics and by physics and by no shortage of myopia in the kinds of relationships a gazetteer might allow.

The hope and the goal for **Who's On First (https://whosonfirst.mapzen.com)** is that the present has afforded us the opportunity to leap-frog many of those limitations. We live in a time when much of the technology around us has taken on a foul-smelling odor but I choose to believe that it is still possible to harness these tools in the service of a common good.

The hope and the goal of Who's On First is not that everyone should have to, or even want to, add their names and stories to our gazetteer but that they *may be able do so* when they choose to.

The hope and the goal is in the chance to allow people the agency to name the places that define their own stories, and in doing so to give them a shared weight and mass in the Universe, without needing to erase someone else's telling in the process.

Thank you.

· 17 October 2017 ·

Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

Explore the world of form with Morphology

cartography (/tag/cartography) tangram (/tag/tangram)

The following words and images were presented at a lightning talk about the Visual Language in Cartography at the ***2017 NACIS Conference (http://nacis.org/)*** *in Montreal.*

What is the visual language of cartography? Let's explore this question through the medium of drawing. After all, it is this abstract representation of place onto a surface of fewer dimensions that the act of cartography entails.

What is this act of drawing a map? Why do we create? What is the mark and its connection to place?

These are questions I like to think about. These are guiding questions which help me to discover, learn and think about what I do everyday as a cartographer.

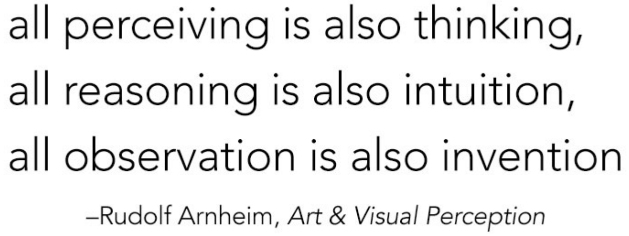
Let's think of drawing a map as an extension of the human experience of the external world onto a two-dimensional surface of paper or pixel display. The drawing is a visual experience.

any visual event is a form with content

–Donis A. Dondis from *A Primer of Visual Literacy*

This visual experience can be composed of color, tone, texture, dimension, scale, proportion and the overall composition that results into a coherent whole. It is complex, it involves layers upon layers of forms, symbols, structure upon structure, all coexisting with one another.

It is a visual language exchange between the creator and observer. The cartographer creates the visual language, the observer deciphers and learns this language.



all perceiving is also thinking,
all reasoning is also intuition,
all observation is also invention

–Rudolf Arnheim, *Art & Visual Perception*

There is a visual intelligence that takes place in this exchange. It is the perception of these layered visual forms and elements and the combinations of which contain multiple layers of meaning. The translation of the visual experience of space to a flat surface involves the abstraction and reduction of reality from an experience of place and of the physical phenomenal world.

Forms

What are these forms? How are they shaped? How do they form? Let’s investigate.

What are the forms that we frequently draw when we draw maps?

In the book *Timeless Way of Building*, Christopher Alexander talks about the patterns of activity that take place in space. According to him these patterns are a result of culture.

He writes, *"…of course the patterns vary from place to place, from culture to culture, from age to age, they are all man-made, they all depend on culture. But still, in every age and every place the structure of our world is given to it, essentially, by some collection of patterns which keeps repeating over and over again."*

> "Each of these patterns is a morphological law, which establishes a set of relationships in space"

…"these patterns have enormous power and depth; they have the power to create an almost endless variety, they are so deep, so general, they combine in millions upon millions of different ways."

In D'Arcy Wentworth Thompson's book *On Growth and Form*, in his study of organic form, the author defines form as:

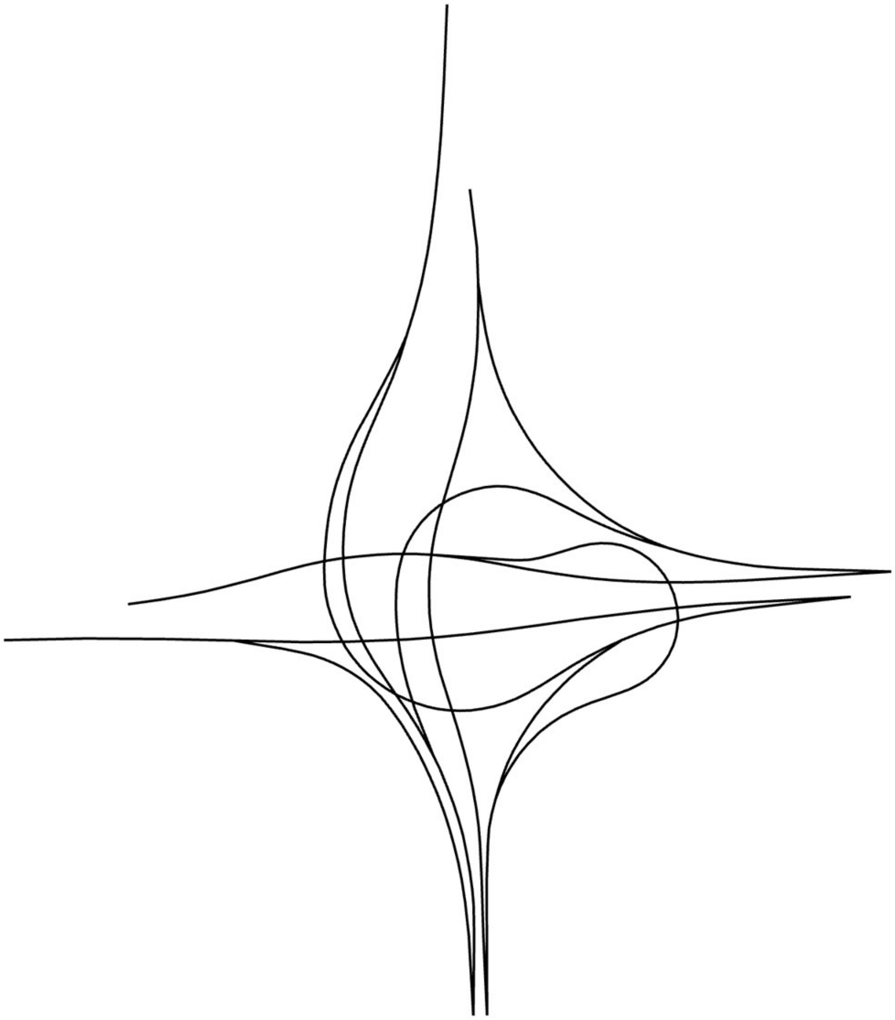
> "the form, then, of any portion of matter, whether it be living or dead, and the changes of form which are apparent in its movements and in its growth, may in all cases alike be described as due to the action of force. In short, the form of an object is a **diagram of forces**."

Lines

Let's have a look at these forms. We will investigate the drawing of these forms with lines. Let's think of lines as the material. The visual experience of the form is affected by the type of material we work with.

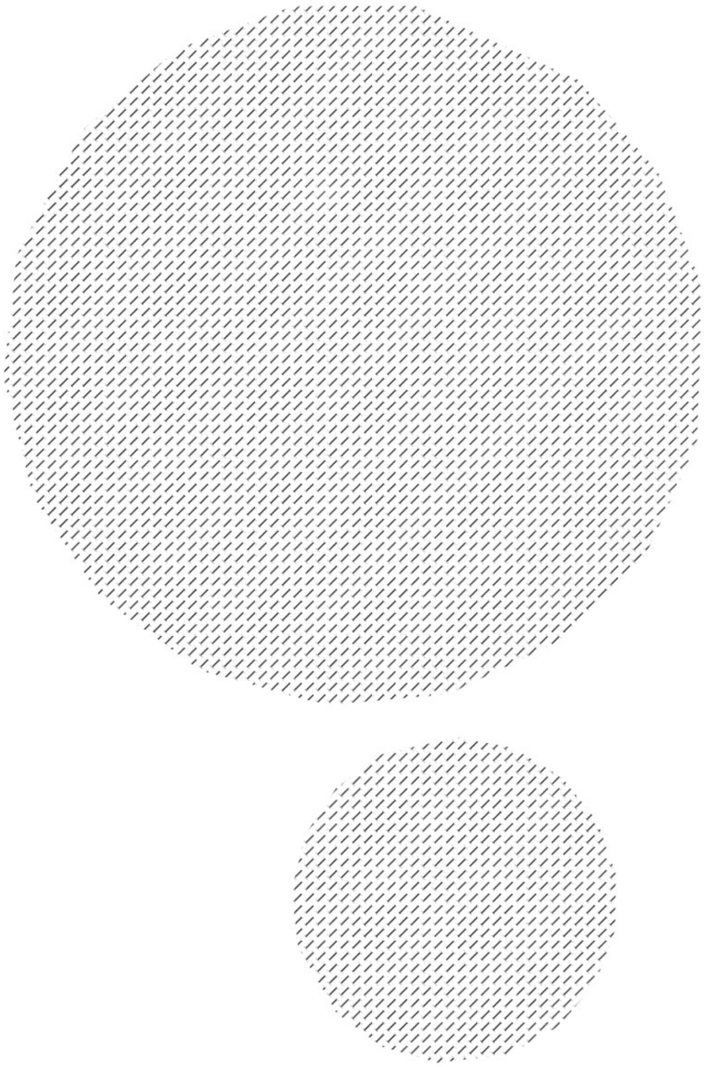
Lines are the most elemental and versatile material we can work with.

With lines we can articulate form, shape and movement.



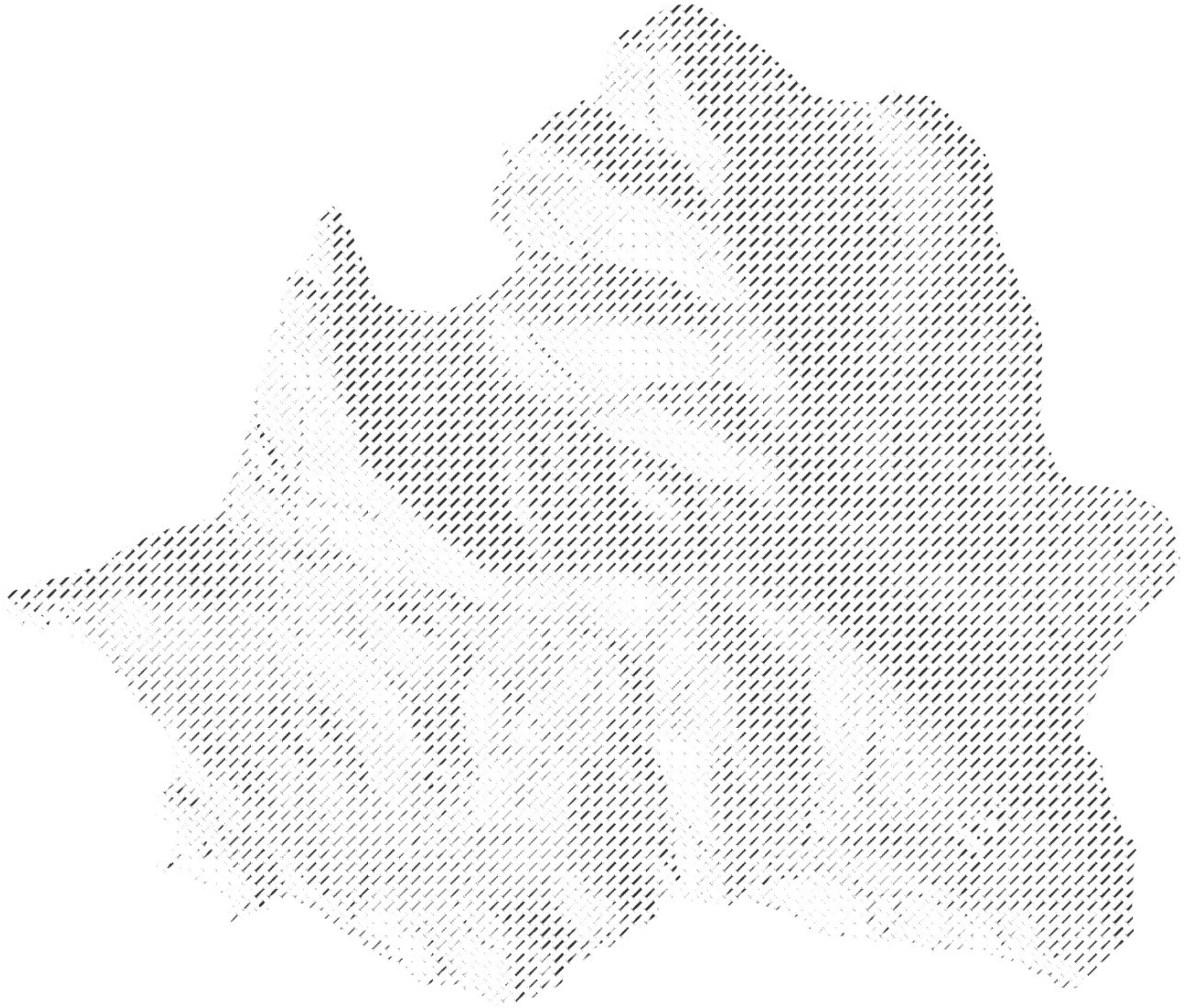
Highway ramps in Shanghai | View the ***map (https://sensescape.github.io/morphology/#lat=31.1587&lng=121.6437&z=16&form=links)***

We can repeat them to create pattern and texture.



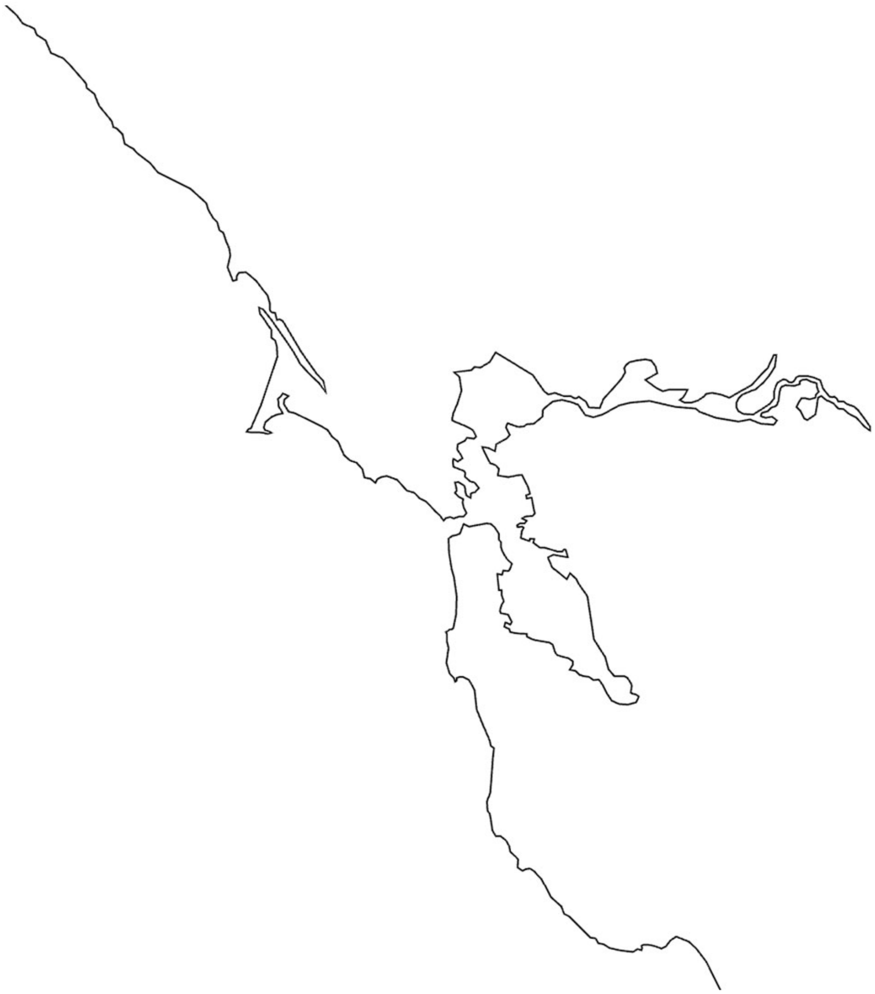
Farms in Nevada | *View the* ***map (https://sensescape.github.io/morphology/#lat=37.5755&lng=-117.9524&z=16&form=farms)***

With line weights we can give detail and dimension.

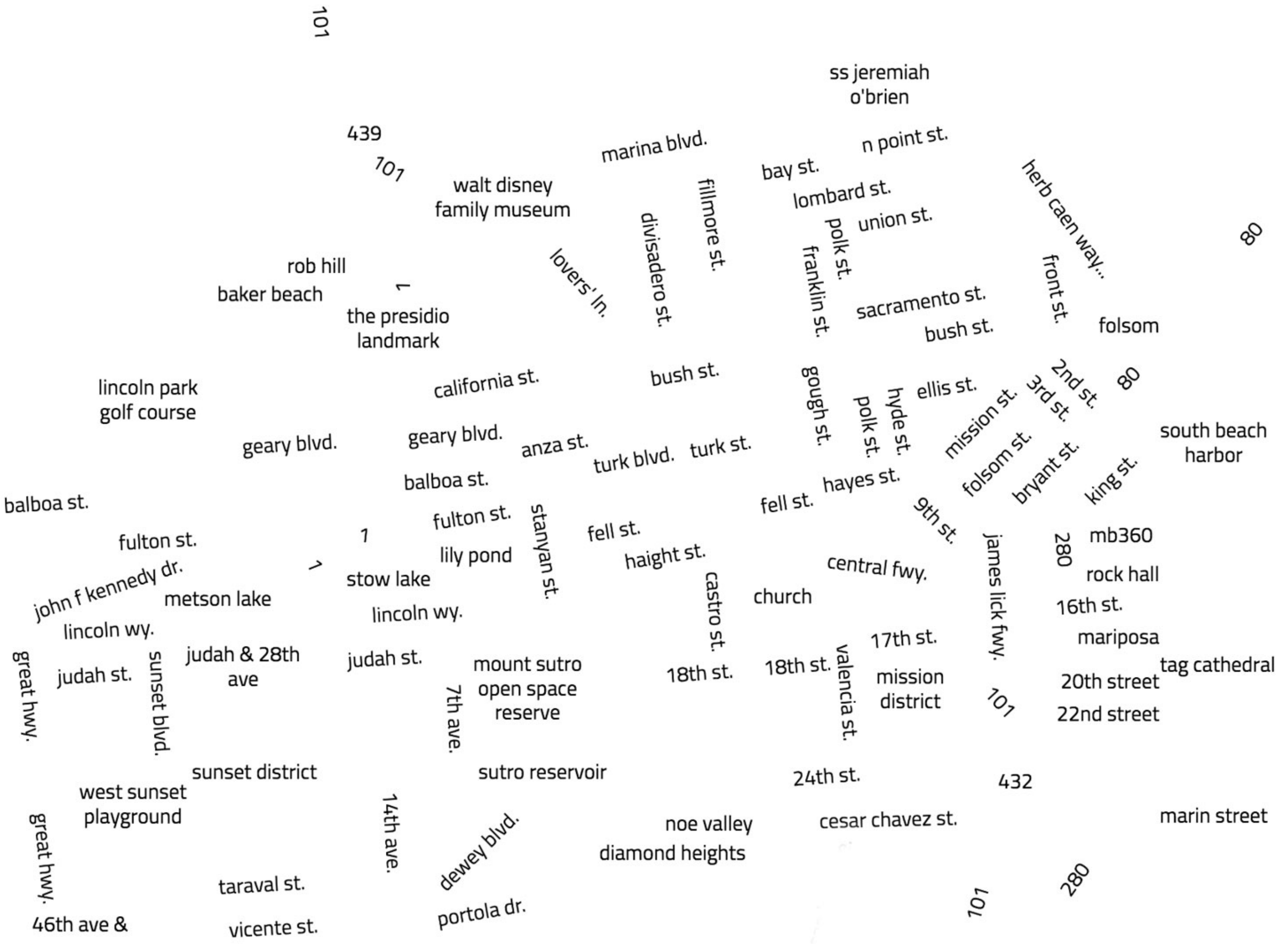


Angel Island | *View the* ***map (https://sensescape.github.io/morphology/#lat=37.8628&lng=-122.4328&z=15&form=green)***

We can create edges and boundaries.



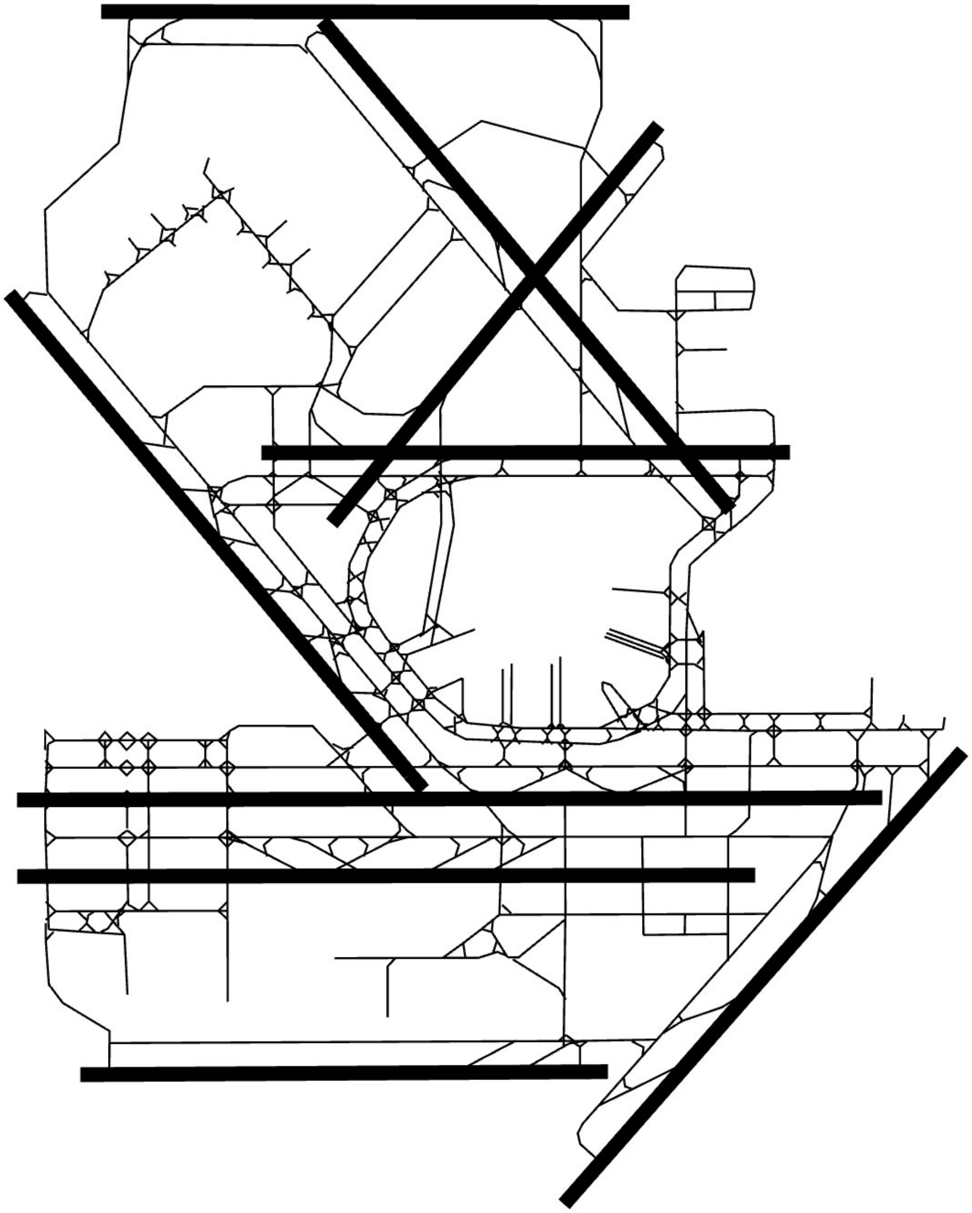
And with specific configurations they become recognizable, decipherable symbols.


101
ss jeremiah o'brien
439
n point st.
marina blvd.
bay st.
101
walt disney family museum
fillmore st.
lombard st.
herb caen way...
union st.
polk st.
80
divisadero st.
lovers' ln.
rob hill
franklin st.
front st.
baker beach
1
the presidio landmark
sacramento st.
bush st.
folsom
california st.
bush st.
gough st.
ellis st.
2nd st.
80
lincoln park golf course
3rd st.
hyde st.
polk st.
mission st.
geary blvd.
geary blvd.
anza st.
south beach harbor
turk blvd.
turk st.
folsom st.
bryant st.
balboa st.
king st.
hayes st.
balboa st.
fell st.
fulton st.
stanyan st.
9th st.
fell st.
fulton st.
1
mb360
lily pond
haight st.
280
1
central fwy.
john f kennedy dr.
stow lake
rock hall
castro st.
james lick fwy.
metson lake
church
16th st.
lincoln wy.
lincoln wy.
17th st.
mariposa
judah & 28th ave
judah st.
mount sutro open space reserve
valencia st.
tag cathedral
great hwy.
judah st.
sunset blvd.
18th st.
18th st.
mission district
20th street
7th ave.
101
22nd street
sunset district
sutro reservoir
24th st.
432
west sunset playground
14th ave.
marin street
great hwy.
dewey blvd.
noe valley
cesar chavez st.
diamond heights
taraval st.
101
280
portola dr.
46th ave &
vicente st.

San Francisco road labels | *View the* ***map*** ***(https://sensescape.github.io/morphology/#lat=37.7788&lng=-122.4426&z=13&form=labels)***

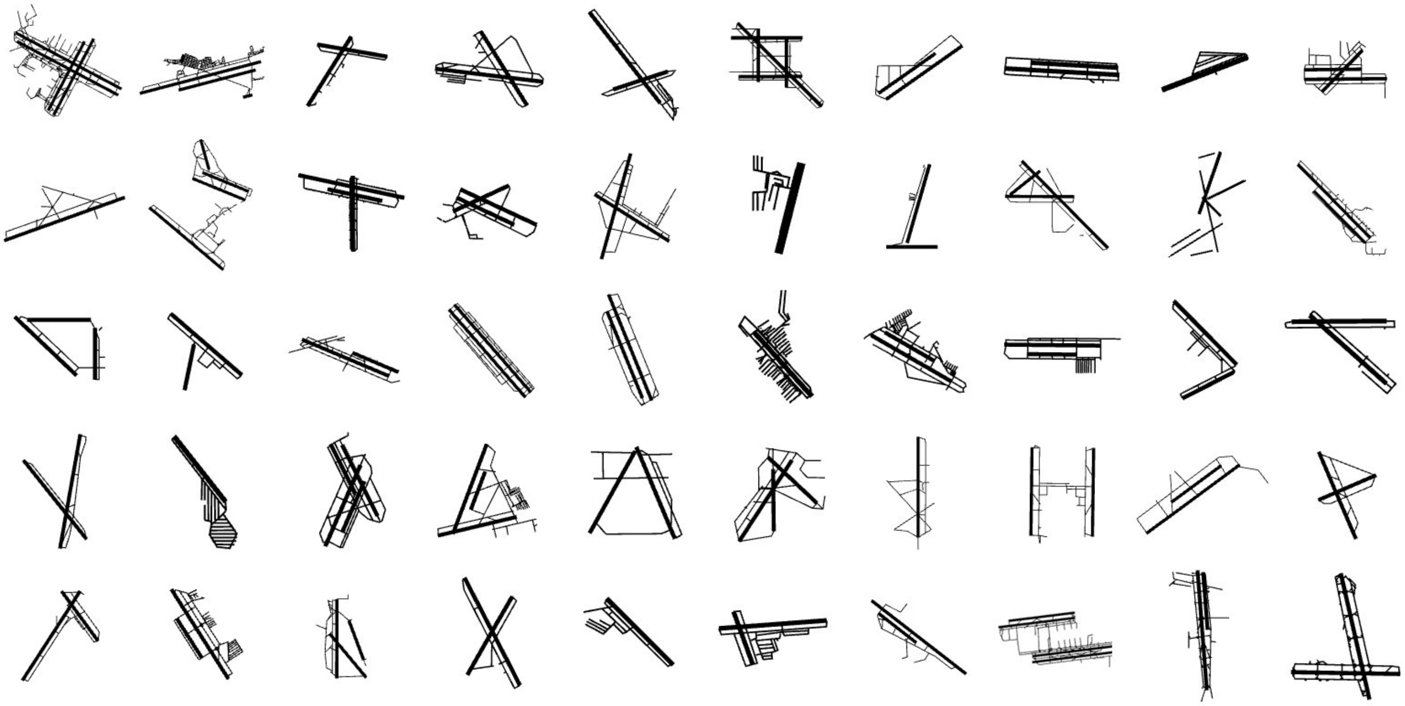
Since most of us are city-dwellers, let’s look at city-specific forms today. Let’s start with airports.

Airports

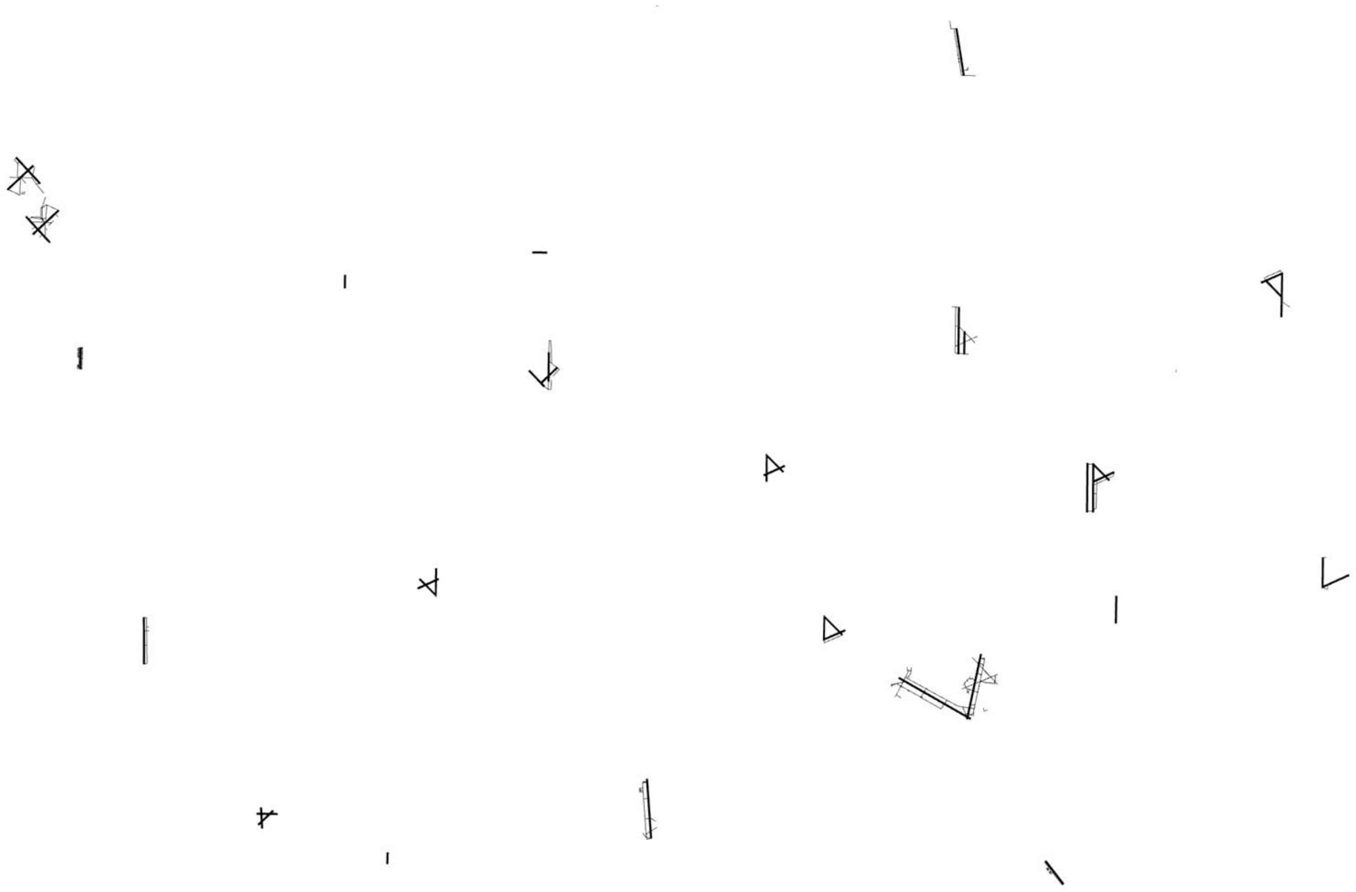


ORD | View the ***map (https://sensescape.github.io/morphology/#lat=41.9792&lng=-87.9050&z=13.7000&form=airports)***

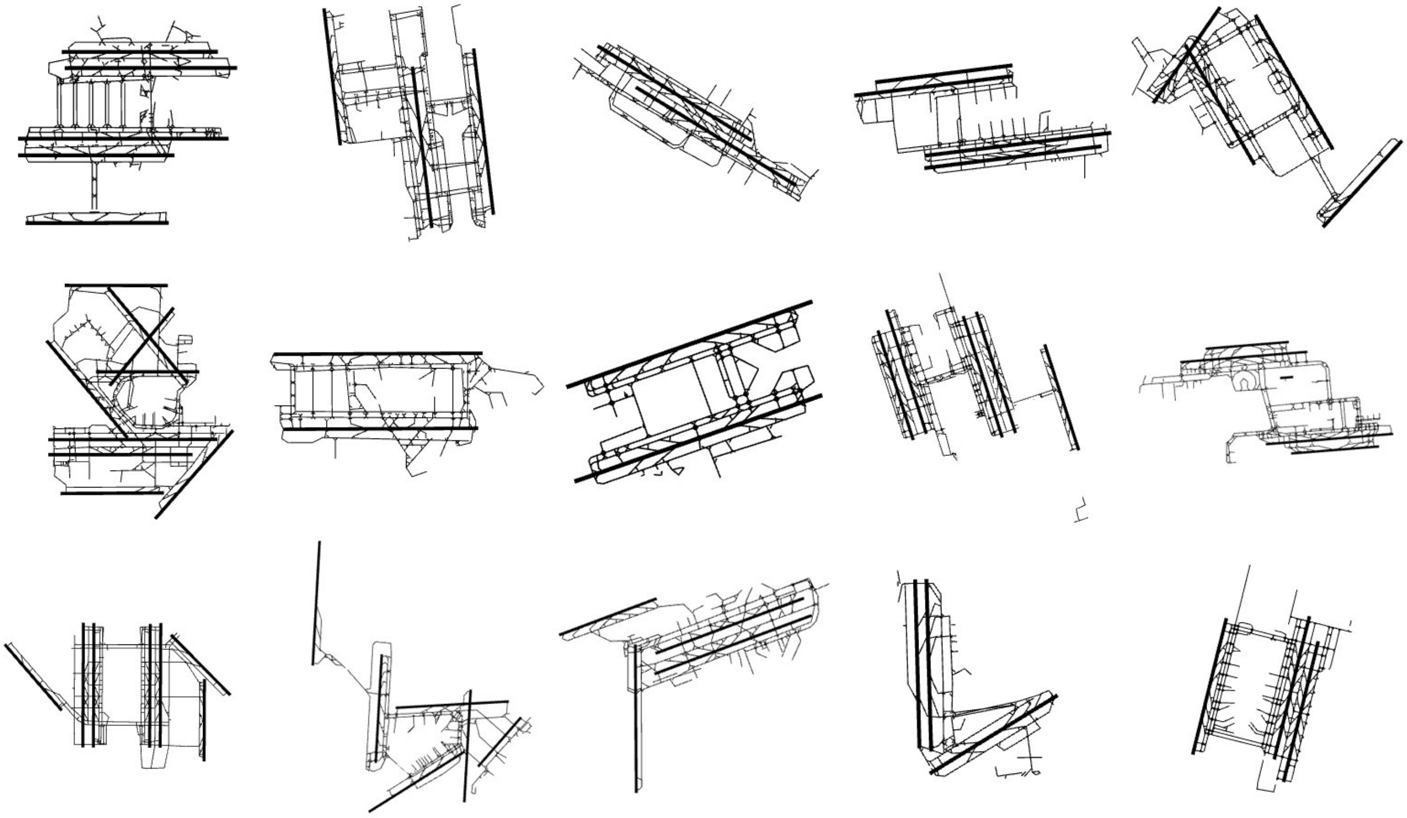
Observe airport runways and taxiways in California below:



A low zoom view of airports in Florida by Pensacola.

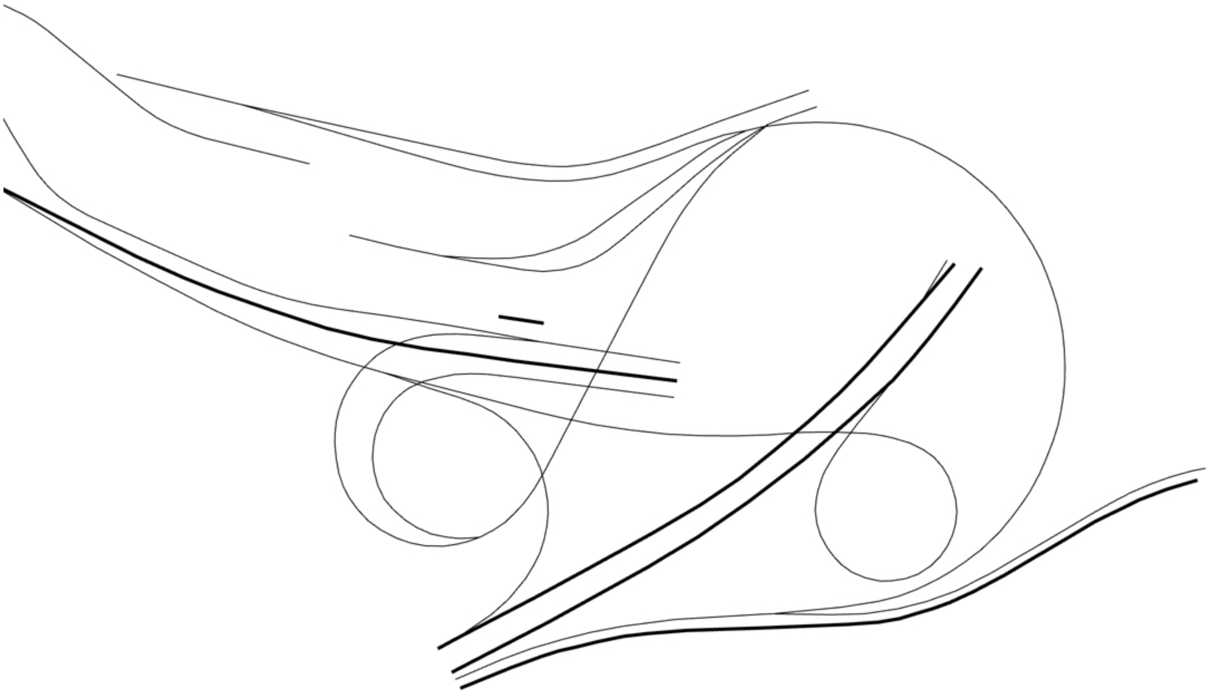


Airports in Florida | View the ***map (https://sensescape.github.io/morphology/#lat=30.5806&lng=-86.7404&z=11&form=airports)***

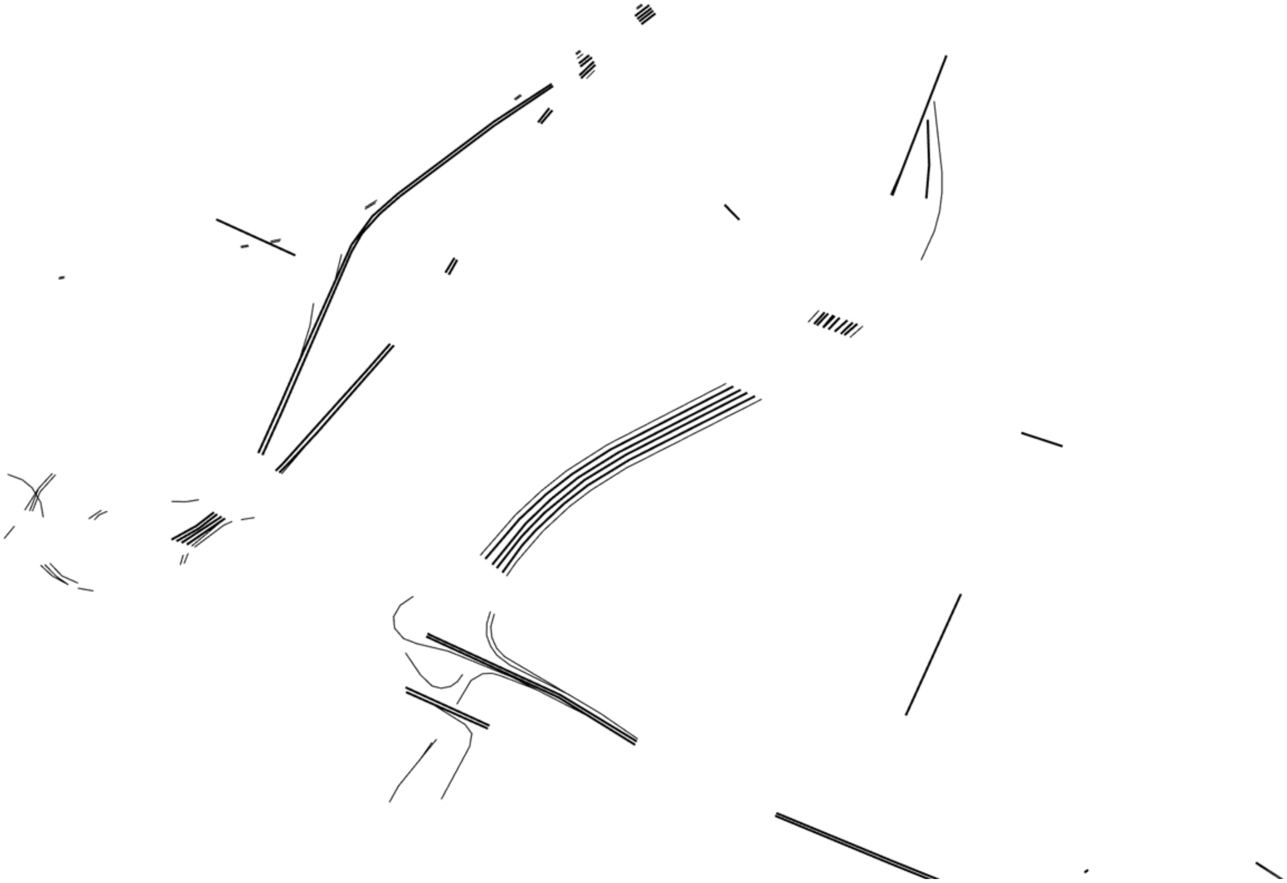


Above are 15 of the busiest airports in the world. Can you guess which cities they belong to?

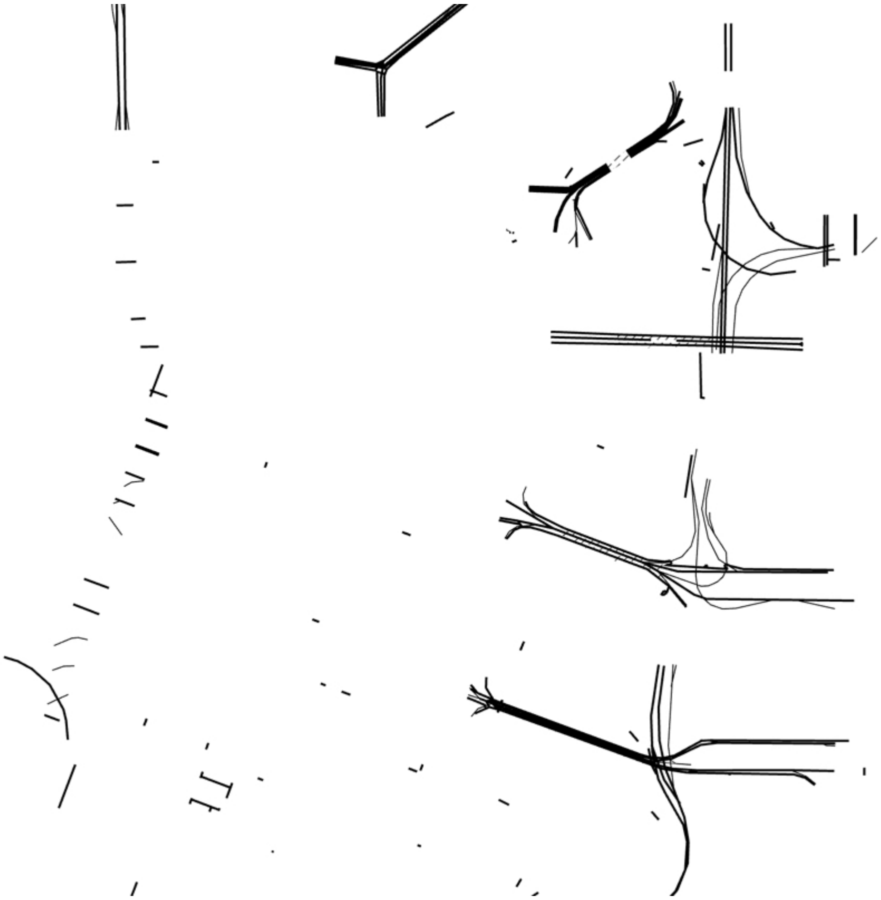
Bridges



Bridges in Alexandria | *View the* ***map***
(https://sensescape.github.io/morphology/#lat=38.79381&lng=-77.05581&z=17&form=bridges)

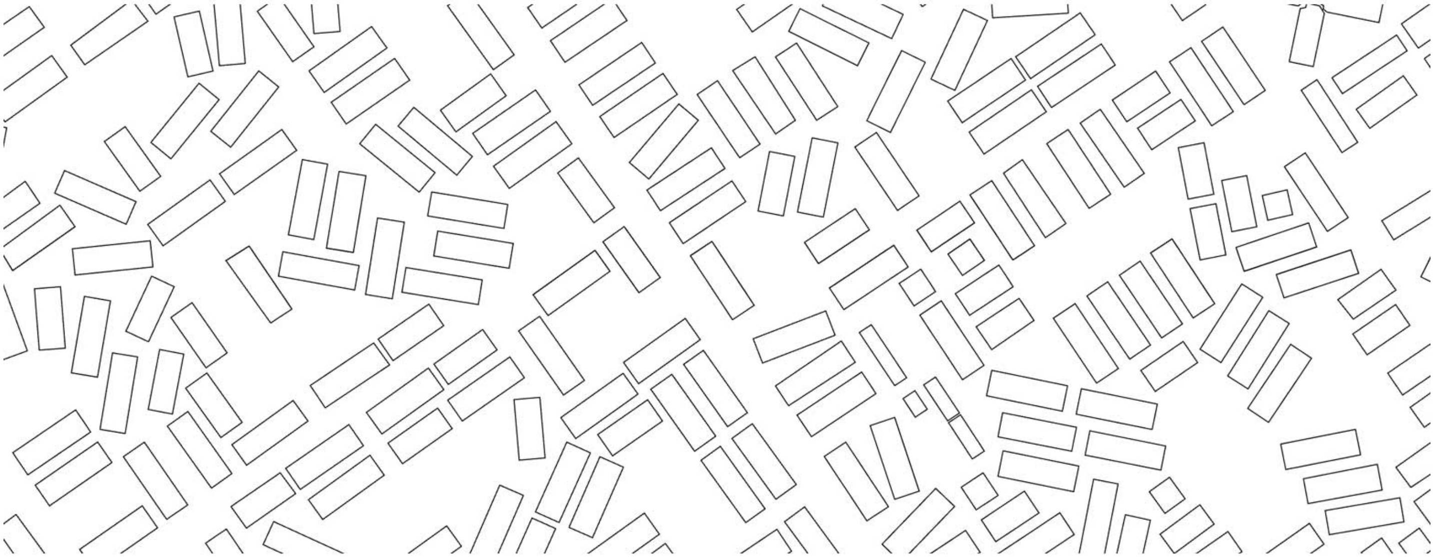


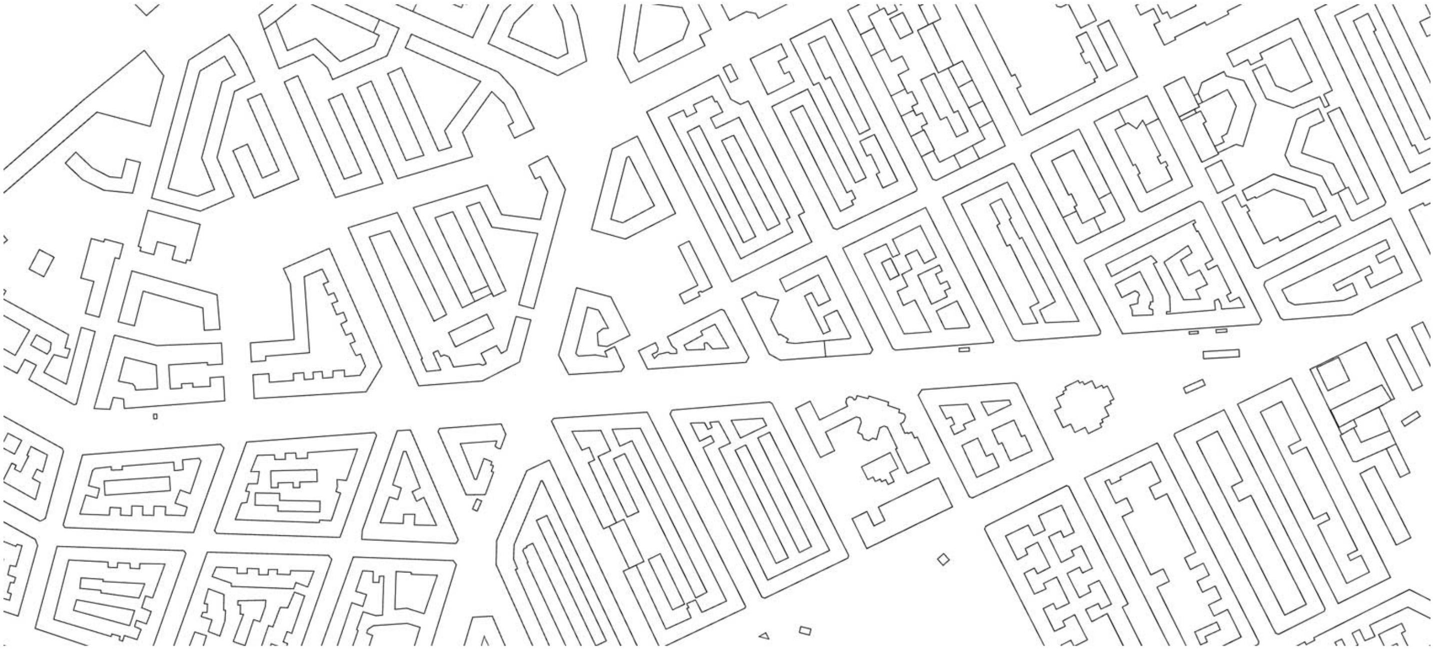
Bridges in Newark | View the ***map (https://sensescape.github.io/morphology/#lat=40.7134&lng=-74.1443&z=15&form=bridges)***



Bridges in Portland | *View the* ***map (https://sensescape.github.io/morphology/#lat=45.5203&lng=-122.6669&z=15&form=bridges)***

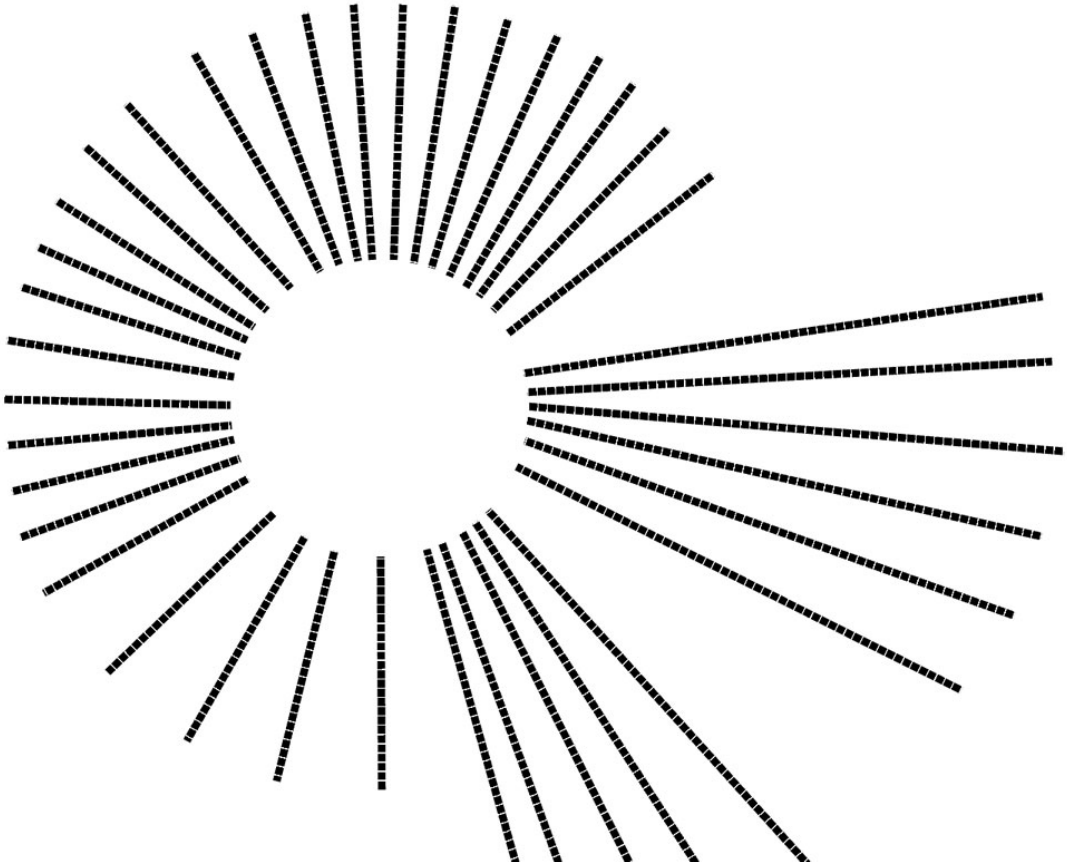
Buildings



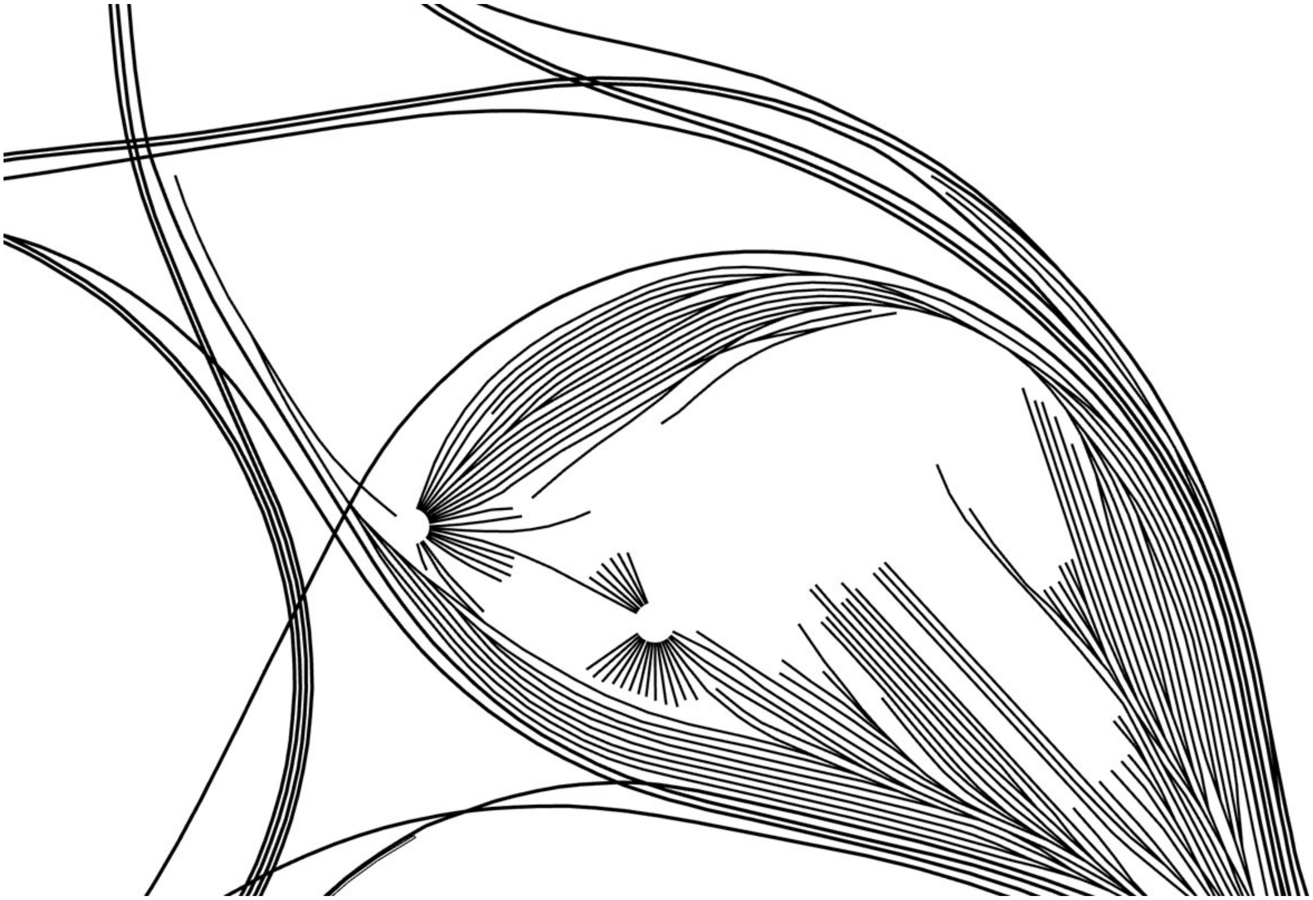


Buildings in ***Brooklyn (https://sensescape.github.io/morphology/#lat=40.64184&lng=-74.00826&z=17&form=buildings)****,* ***Kolkata (https://sensescape.github.io/morphology/#lat=22.48150&lng=88.36739&z=17&form=buildings)****,* ***Los Angeles (https://sensescape.github.io/morphology/#lat=33.76991&lng=-118.08559&z=17&form=buildings)****,* ***Stockholm (https://sensescape.github.io/morphology/#lat=59.34326&lng=18.04132&z=16.2500&form=buildings)****,* ***Manhattan (https://sensescape.github.io/morphology/#lat=40.75810&lng=-73.97171&z=17&form=buildings)***

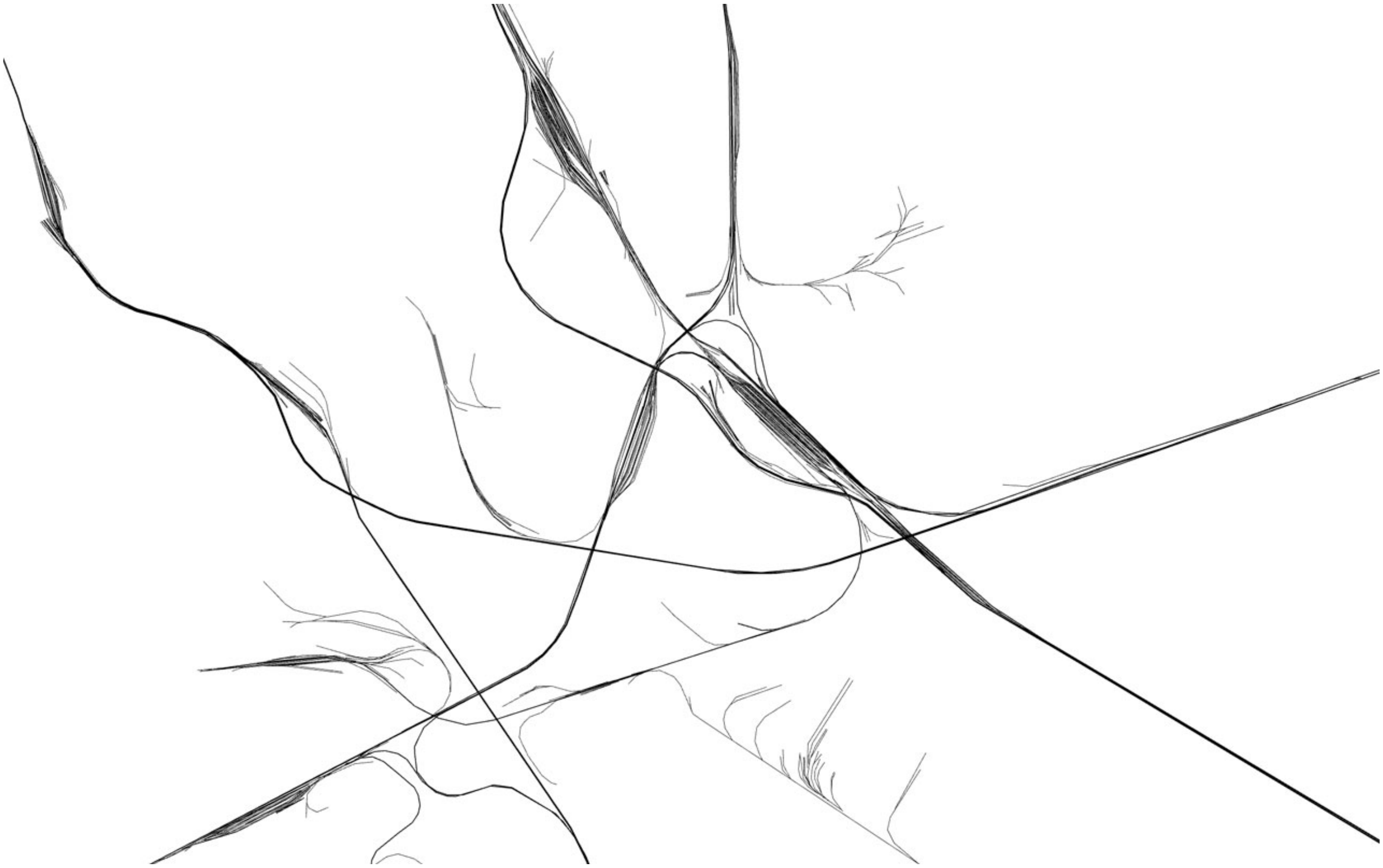
Rail



Rail in Bucharest | *View the* ***map*** ***(https://sensescape.github.io/morphology/#lat=44.46088&lng=26.05699&z=19.5000&form=rail)***

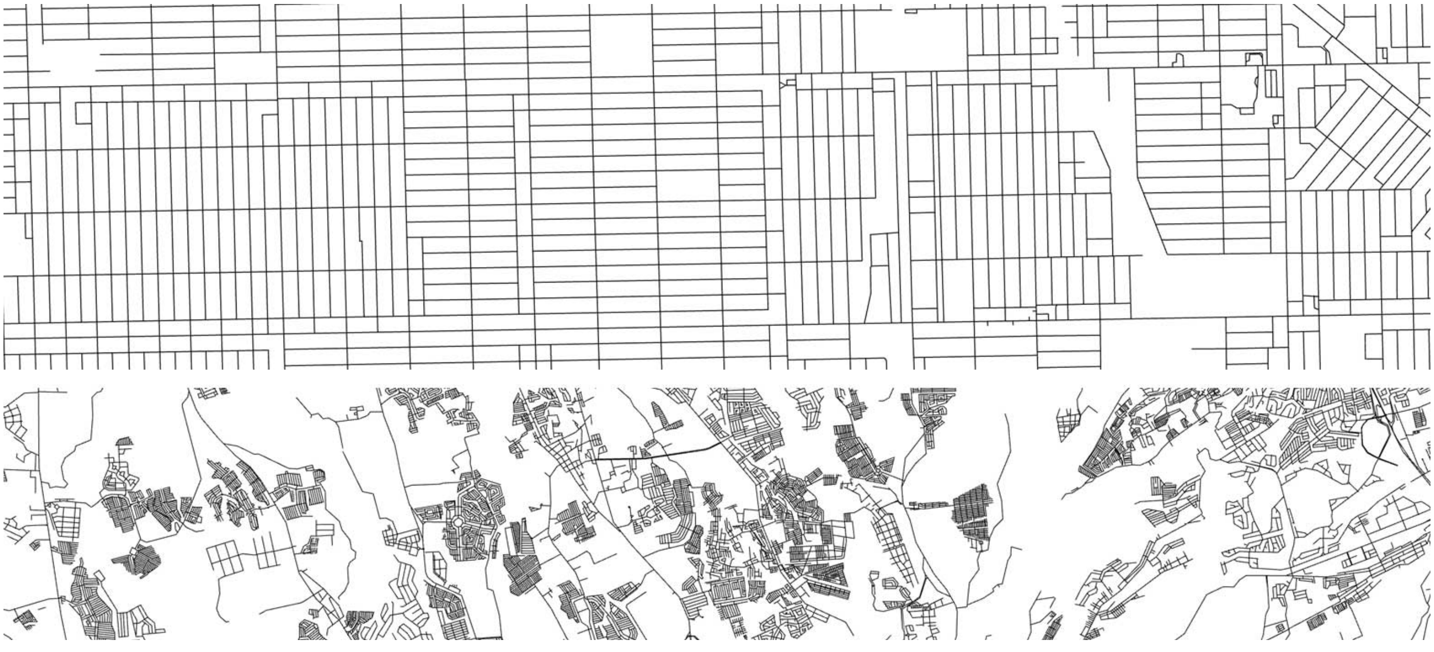


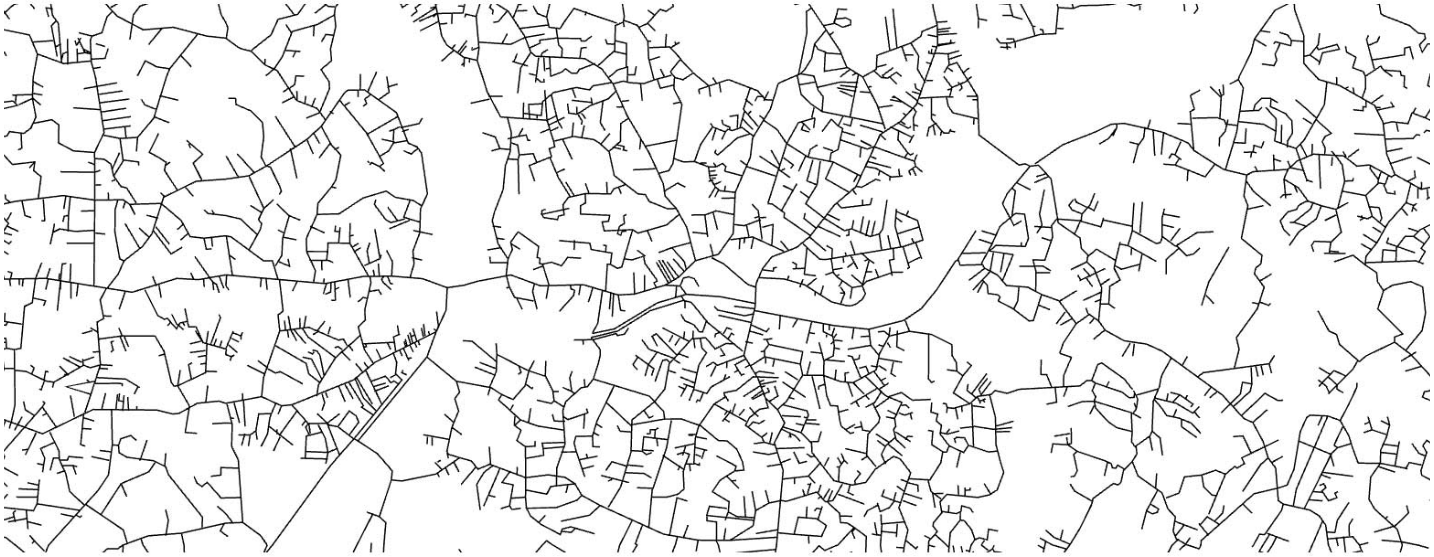
Rail in Cologne | View the ***map (https://sensescape.github.io/morphology/#lat=50.95249&lng=6.94652&z=16.7500&form=rail)***

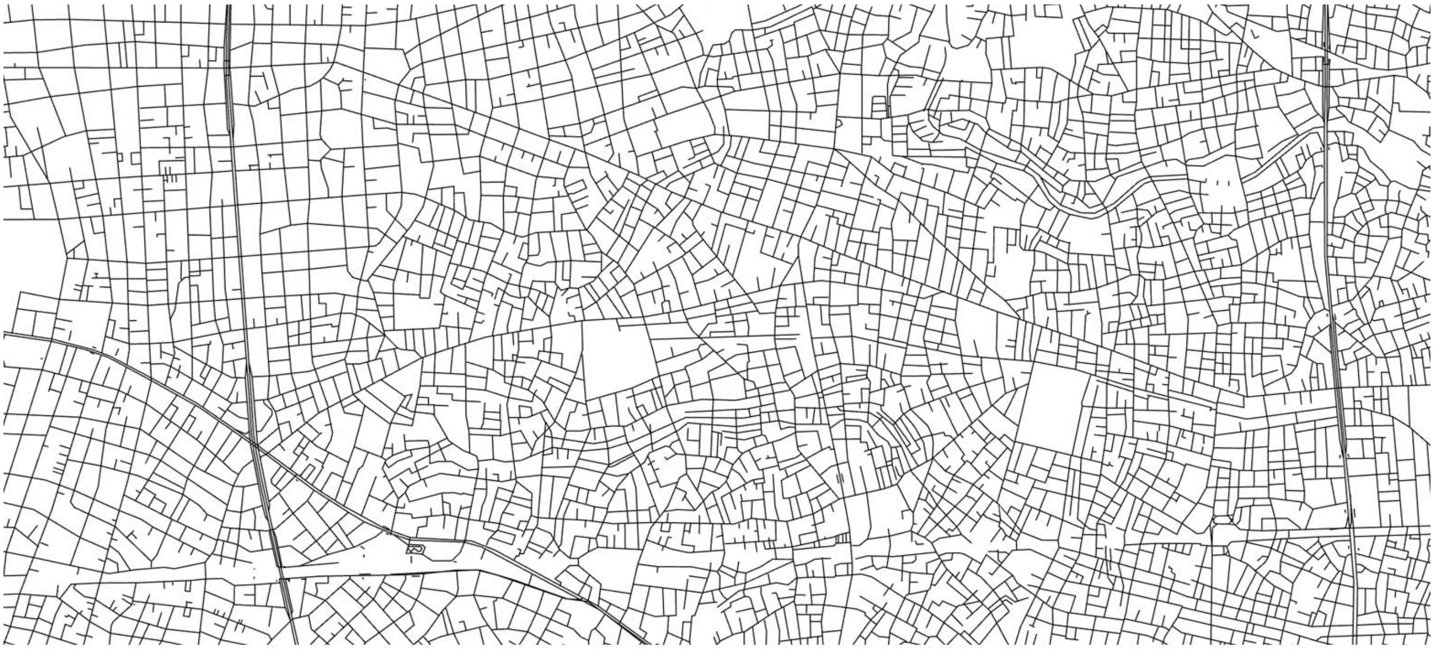


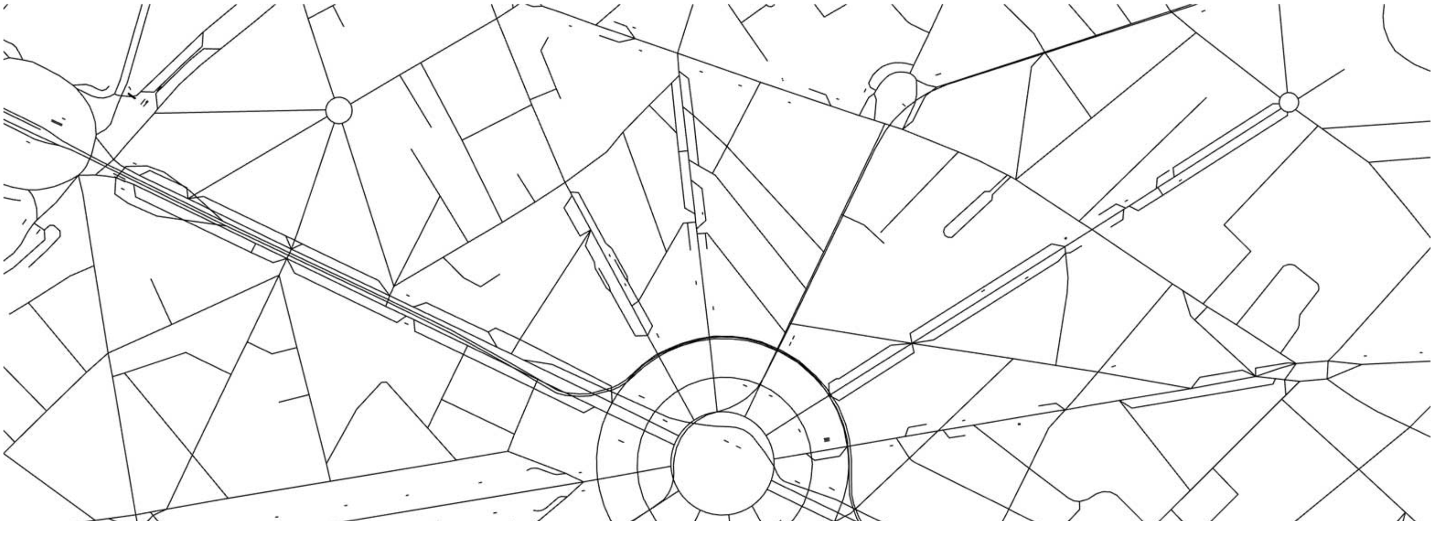
Rail in Moscow | View the ***map (https://sensescape.github.io/morphology/#lat=55.7468&lng=37.7391&z=13.500&form=rail)***

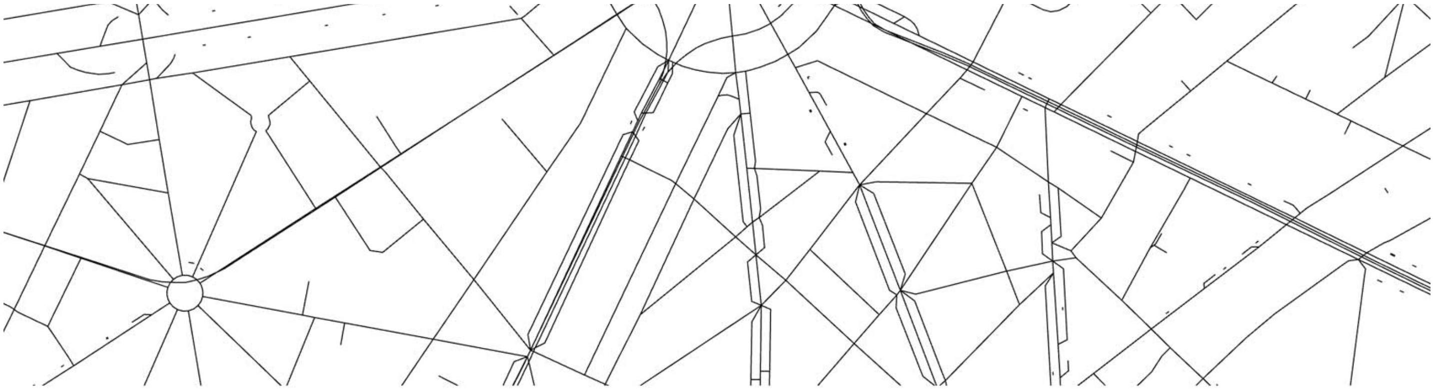
Roads







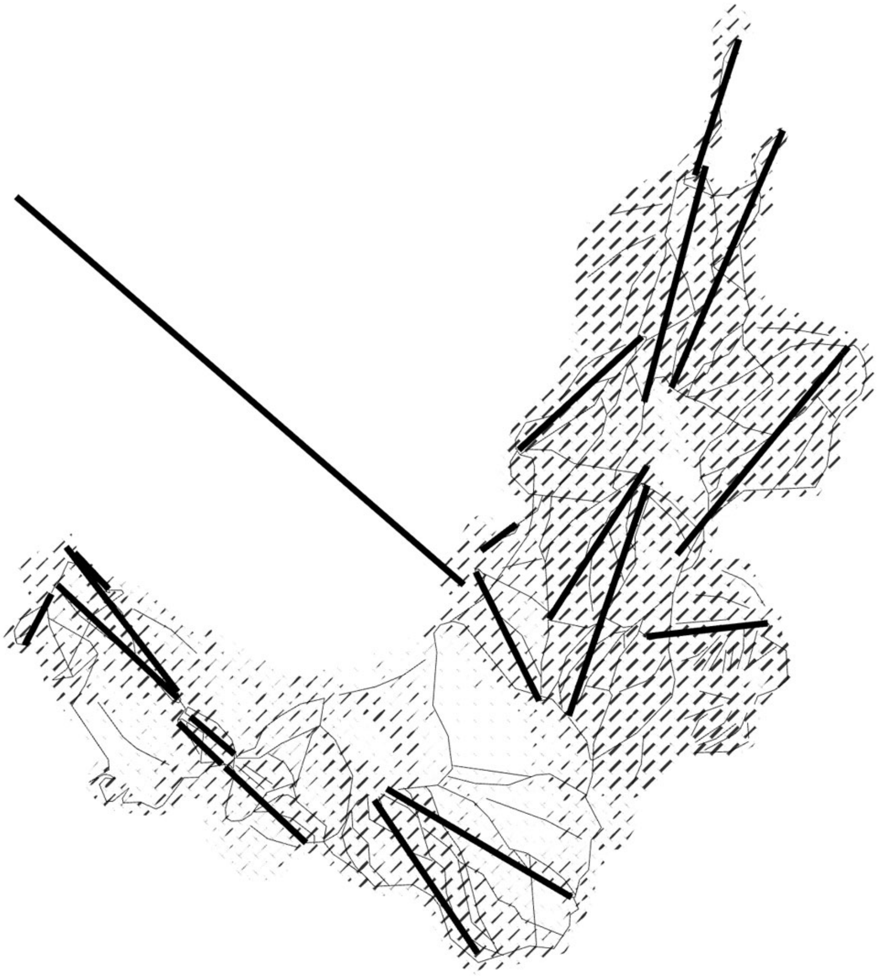




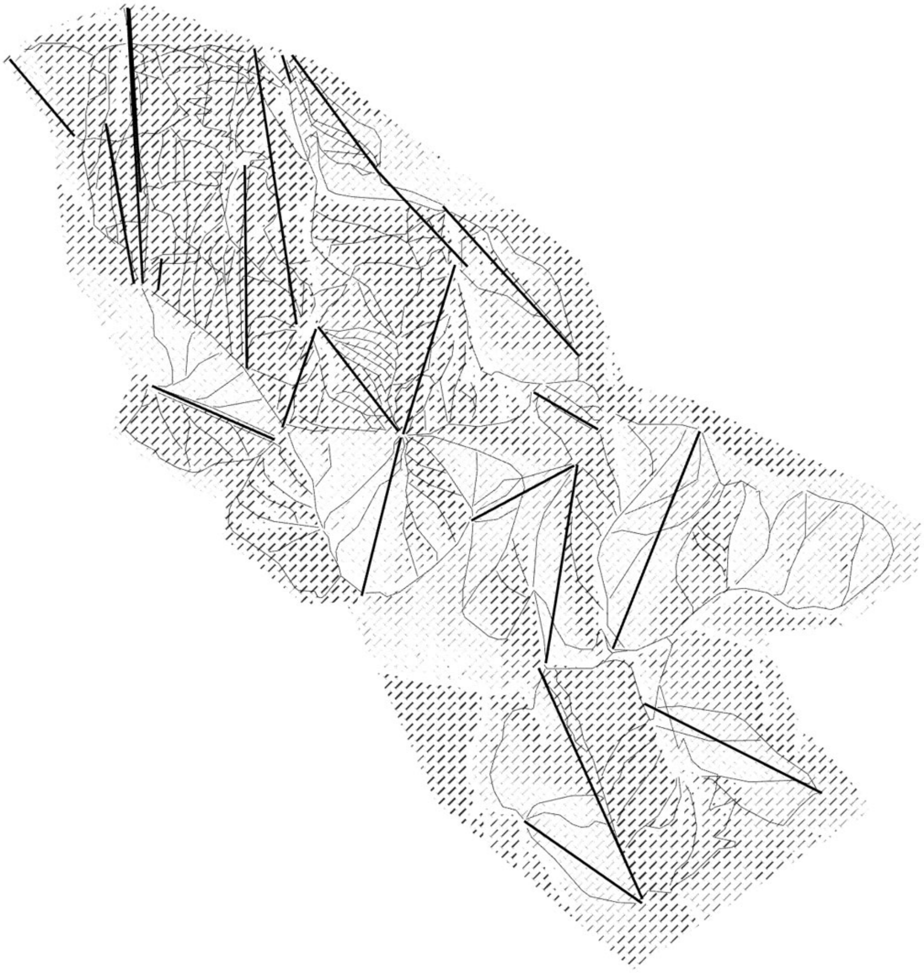
Roads in ***Chicago (https://sensescape.github.io/morphology/#lat=41.9309&lng=-87.7493&z=15&form=roads)****,* ***Dasmarinas (https://sensescape.github.io/morphology/#lat=14.3219&lng=120.9653&z=13&form=roads)****,* ***Colombo (https://sensescape.github.io/morphology/#lat=7.0695&lng=79.9843&z=14&form=roads)****,* ***Tokyo (https://sensescape.github.io/morphology/#lat=35.7094&lng=139.6277&z=15&form=roads)****,* ***Paris (https://sensescape.github.io/morphology/#lat=48.8739&lng=2.2949&z=16&form=roads)***

Ski

*Breckenridge | View the **map** **(https://sensescape.github.io/morphology/#lat=39.6287&lng=-105.8732&z=14.5000&form=ski)***

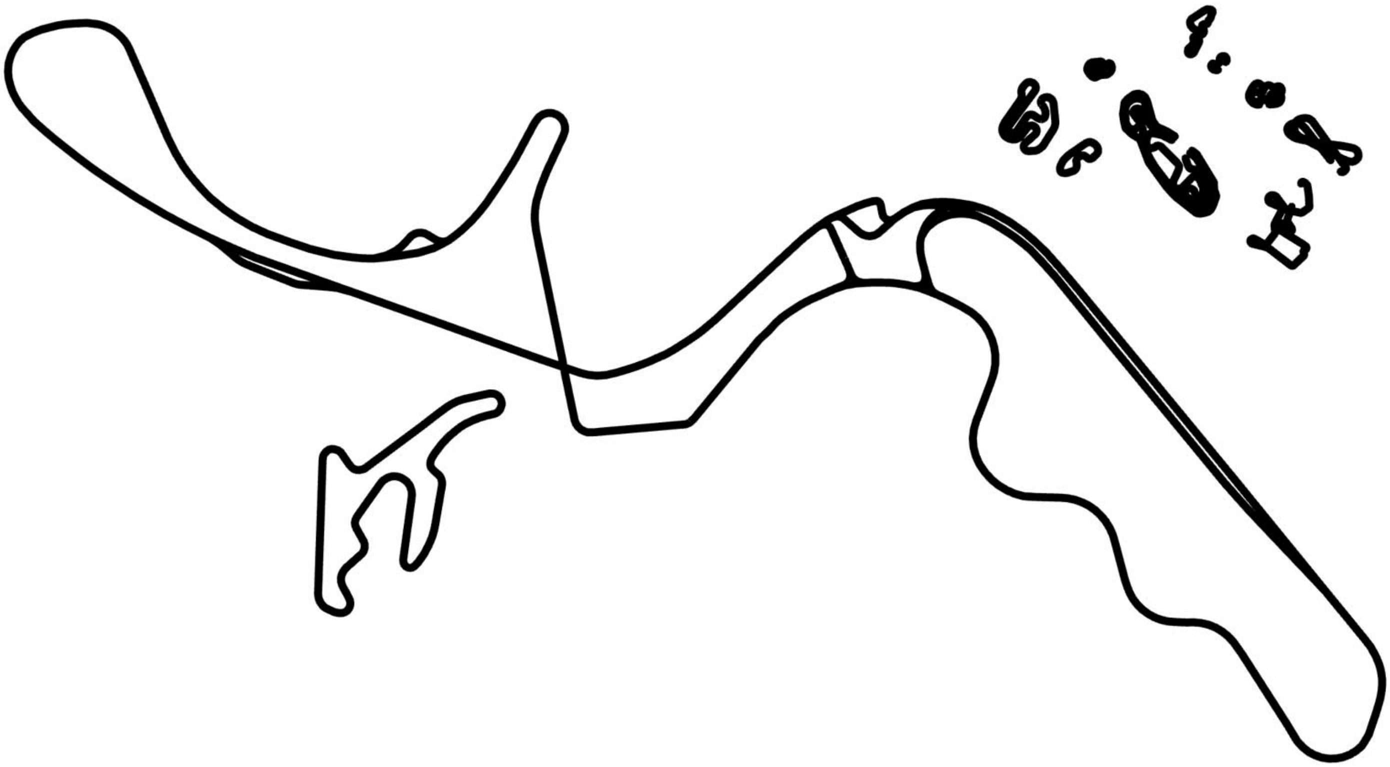


Lake Tahoe | *View the* ***map***
(https://sensescape.github.io/morphology/#lat=38.9397&lng=-119.9114&z=13.7500&form=ski)

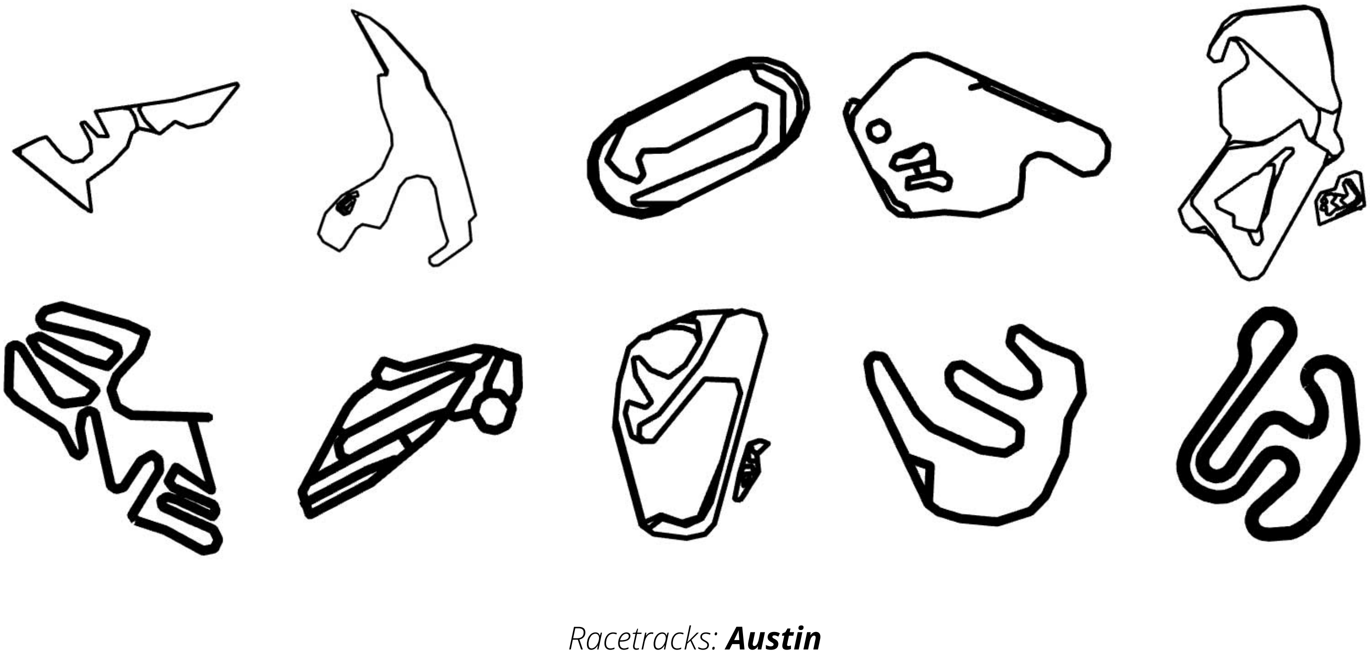


Vail | View the ***map***
(https://sensescape.github.io/morphology/#lat=39.6012&lng=-106.3479&z=13.1000&form=ski)

Racetracks



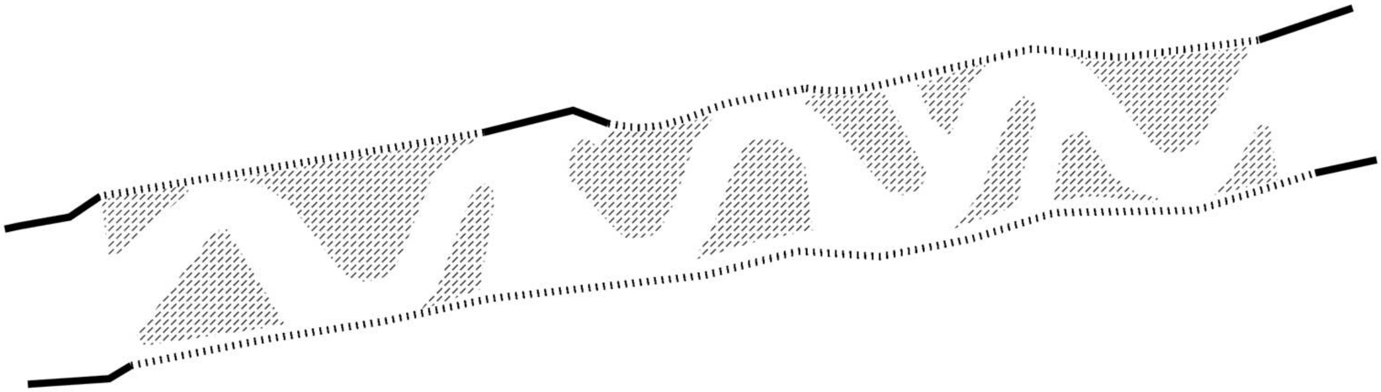
Suzuka | View the ***map (https://sensescape.github.io/morphology/#lat=34.8438&lng=136.5326&z=16&form=racetracks)***



Racetracks: ***Austin (https://sensescape.github.io/morphology/#lat=30.1354&lng=-97.6333&z=16&form=racetracks)****,* ***Francorchamps***

(https://sensescape.github.io/morphology/#lat=50.4375&lng=5.9691&z=15&form=racetracks), ***Homestead (https://sensescape.github.io/morphology/#lat=25.45182&lng=-80.40862&z=17&form=racetracks)***, ***Lime Rock (https://sensescape.github.io/morphology/#lat=41.92755&lng=-73.38363&z=17&form=racetracks)***, ***Milton Keynes (https://sensescape.github.io/morphology/#lat=52.0720&lng=-1.0163&z=15&form=racetracks)***, ***Nantes (https://sensescape.github.io/morphology/#lat=47.19822&lng=-1.64053&z=17.5000&form=racetracks)***, ***Nürburg (https://sensescape.github.io/morphology/#lat=50.32672&lng=6.93251&z=18.2000&form=racetracks)***, ***Sao Paolo (https://sensescape.github.io/morphology/#lat=-23.70139&lng=-46.69669&z=16.2000&form=racetracks)***, ***Rosamond (https://sensescape.github.io/morphology/#lat=34.8753&lng=-118.2636&z=16&form=racetracks)***, ***Suzuka (https://sensescape.github.io/morphology/#lat=34.8438&lng=136.5326&z=16&form=racetracks)***

Explore The World Through The Morphology Map



drawing/map/tool | Lombard Street in San Francisco | View the ***map (https://sensescape.github.io/morphology/#lat=37.80210&lng=-122.41881&z=20&form=green)***

Morphology is a **drawing/map/tool (https://sensescape.github.io/morphology/#lat=37.6193&lng=-122.3763&z=14)** created for the study of forms in cartography. It is an exploration of form through the medium of lines. As seen from the examples above, forms can be viewed in isolation.

The following forms available for view currently are *airports, bridges, buildings, canals, city walls, earth, farms, green areas, industrial areas, libraries, links, parking, piers, places of worship, quarries, racetracks, rail, roads, schools, ski, tunnels and water*. A few combinations are available as well, such as *bridges* + *water*, *canals* + *water*. A *labels only* view is also available. Specific locations of isolated forms are bookmarked for your use.

See an empty map? Some forms start at later zoom levels. Use the form drop down menu to change back to Morphology's default view to see where you are.

Thank you.

Answers to 15 busiest airports from left to right: ATL, PEK, DXB, LAX, HND, ORD, LHR, HKG, PVG, CDG, DFW, AMS, FRA, IST, CAN

REFERENCES:

A Primer of Visual Literacy (https://www.amazon.com/Primer-Visual-Literacy-Donis-Dondis/dp/0262540290) by Donis A. Dondis, **Art and Visual Perception (https://www.amazon.com/Art-Visual-Perception-Psychology-Creative/dp/0520243838)** by Rudolf Arnheim, **The Timeless Way of Building (https://www.amazon.com/Timeless-Way-Building-Christopher-Alexander/dp/0195024028)** by Christopher Alexander, **On Growth and Form (https://www.amazon.com/Growth-Form-DArcy-Wentworth-Thompson/dp/0521437768)** by D'Arcy Wentworth Thompson

· 19 October 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

© 2017 Mapzen

OSMLR hits a "mile marker" (and joins AWS Public Datasets)

mobility (/tag/mobility)

tldr; Mapzen is collaborating with Amazon Web Services to release OSMLR segments through their Public Datasets program. Use OSMLR segments and IDs to attach your dynamic data to almost any stretch of roadways in OpenStreetMap, anywhere in the world.

On a long road trip, one of the most dependable ways of passing the time is watching the mile markers: Another mile down, another mile closer to your destination. Mile markers are useful if you get a flat tire and need to describe your location over the phone. And they're even more useful to the public agencies and contractors that maintain roadways. Mile markers serve as a *linear referencing* system, which can be used to catalog stretches of pavement needing repair or locations along the roadway where collisions have occurred. Linear referencing is also how roadways speeds and other data about traffic conditions are collected and shared.

OSMLR is what we call the linear referencing system that we've built on top of OpenStreetMap. OSM has great information about roads around the world and their interconnections, but it lacks the means to give a stable identifier to a stretch of roadway. OSMLR provides a stable set of numerical IDs for every 1 kilometer stretch of roadway around the world. In urban areas, OSMLR IDs are attached to each block of roadways between significant intersections.

Just as there are many uses of the mile marker signs along a highway, there are many possible uses for OSMLR:

- We developed OSMLR to associate historical traffic speeds with roadway segments for **our work with the World Bank on the Open Traffic platform (/blog/announcing-open-traffic/)**.
- OSMLR is an effective means for associating roadways with dynamic data that doesn't belong in a static map dataset like OSM. For example: planned construction, parade routes, or on-street parking usage statistics.
- OSMLR can be used to associated OSM roadways with proprietary or confidential data that doesn't belong in OSM.
- OSMLR provides all the key ingredients to match OSM roadways with road maps from another base map provider. In the future, once more tooling is available, it it should be straightforward to transfer data associated with OSM to another base map, for example the U.S. Department of Transportation's ARNOLD.

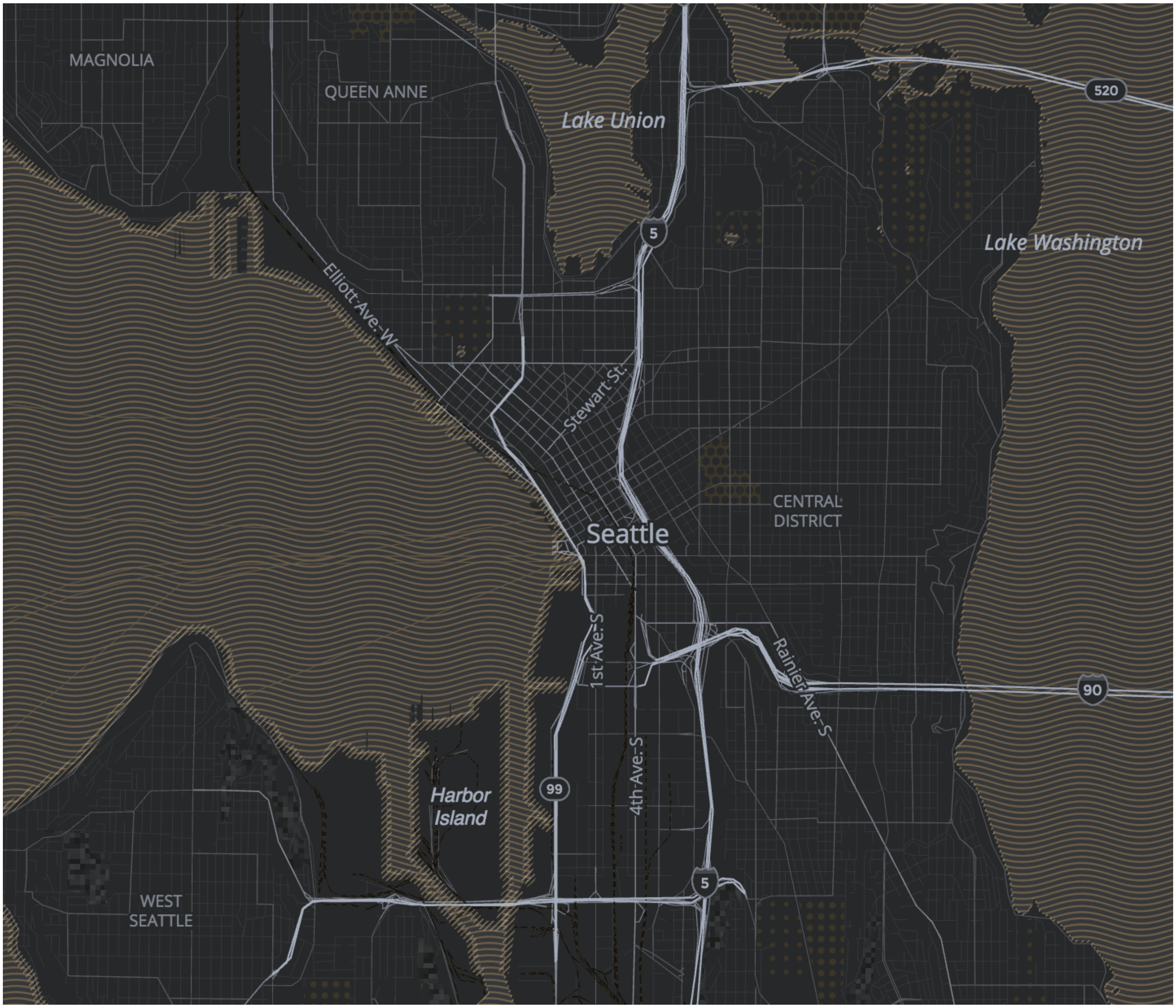
Since OSMLR has such a wide range of uses and users, we're now collaborating with **Amazon Web Services** (AWS) to release the worldwide OSMLR segment tileset through the **AWS Public Datasets (https://aws.amazon.com/public-datasets/)** program. OSMLR segments will be freely available through Amazon S3, indefinitely. AWS users wishing to run advanced analytics on OSMLR segments will find it very speedy — the entire OSMLR planet tileset is already within fast reach on the AWS network. Mapzen and AWS have already worked together to release **terrain data (/blog/terrain-tile-service)** through the Public Datasets program, and we've been pleased with all the options for visualization and analysis that it's opened up.

Now that OSMLR is in the AWS Public Datasets program, we've moved OSMLR segments to their new canonical home in an S3 bucket called `osmlr`. You can **browse listings of all the available tile files (https://s3.amazonaws.com/osmlr/listing.html)**. To download subsets of OSMLR tiles, **we've created a set of shell and Python scripts (https://github.com/opentraffic/osmlr/tree/master/py)** you can use to fetch tiles within your choice of bounding box:

For example, fetch some tiles for Seattle like so:

```
./download_tiles.sh -122.4426,47.4986,-122.168,47.6894 https://s3.amazonaws.com/osmlr/v1.
```

And you can create **a map of downtown Seattle (https://mapzen.com/tangram/view/?api=22/1139#13.1164/47.6092/-122.3417)** showing three OSMLR tiles: highways from the level 0 tile encompassing Seattle are in red, arterial roads from the level 1 tile are in yellow, and local roads from the level 2 tile are in orange. (For more on the three levels of the OSMLR tile hierarchy, see **this blog post (/blog/osmlr-2nd-technical-preview)**.)


MAGNOLIA
QUEEN ANNE
Lake Union
520
5
Lake Washington
Elliott Ave W
Stewart St
CENTRAL DISTRICT
Seattle
1st Ave S
Rainier Ave S
90
99
4th Ave S
Harbor Island
5
WEST SEATTLE

At Mapzen, we're excited to continue improving OSMLR. We'll be sharing more information soon on the **quarterly update process (https://github.com/opentraffic/osmlr/issues/7)** that will keep OSMLR fresh with the latest OSM edits and additions. We'll also be exposing OSMLR segment IDs through more services hosted on **Mapzen Flex (/pricing)**. More on this in the coming months.

Our collaborators at the **World Bank** are also expanding their use of OSMLR beyond Open Traffic. Expect more from them on their Open Transport Partnership soon.

Want to learn more about OSMLR? Come to our to **talk today at State of the Map US in Boulder (https://2017.stateofthemap.us/program/stable-linear-referencing-system.html)**. Read an **introductory technical blog post (/blog/open-traffic-osmlr-technical-preview)** and a **follow-up post (/blog/osmlr-2nd-technical-preview)**.

We'll be following up more tutorials and documentation on OSMLR soon. In the meantime, **we welcome your questions, comments, ideas, and creations (mailto:hello@mapzen.com)**.

Mile marker picture by ***Daniel Ramirez (https://www.flickr.com/photos/danramarch/8127236893/in/photolist-dobdo2-atmx58-U2j5BF-bEX7o9-dYm9VB-a5z4n4-V1sLrG-U2j3Jx-V4fXtr-aFj4AV-nfZHrW-pQdHJP-ebWPuw-a8xz91-bAbnQa-aQcbZi-9bx36q-qTJZTs-mmGFqD-5mv7xb-8vdutd-b9gfV6-athMnH-dKdkMP-8KsfT7-adiHn8-orstGV-aiqS5g-a3nk6o-bEujiH-9bx3s7-9bx3o5-9btVSx-bRDoDn-7AngyX-oQKftF-9pp15N-qAxcze-7Y488x-ryhHZX-mdjWv-5bPXrT-aQccBV-T9DSXn-cyEAA-NAMiX-pZWK4T-5mdEvS-7AnhAp-qgMYwp)***.

· 20 October 2017 ·

Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

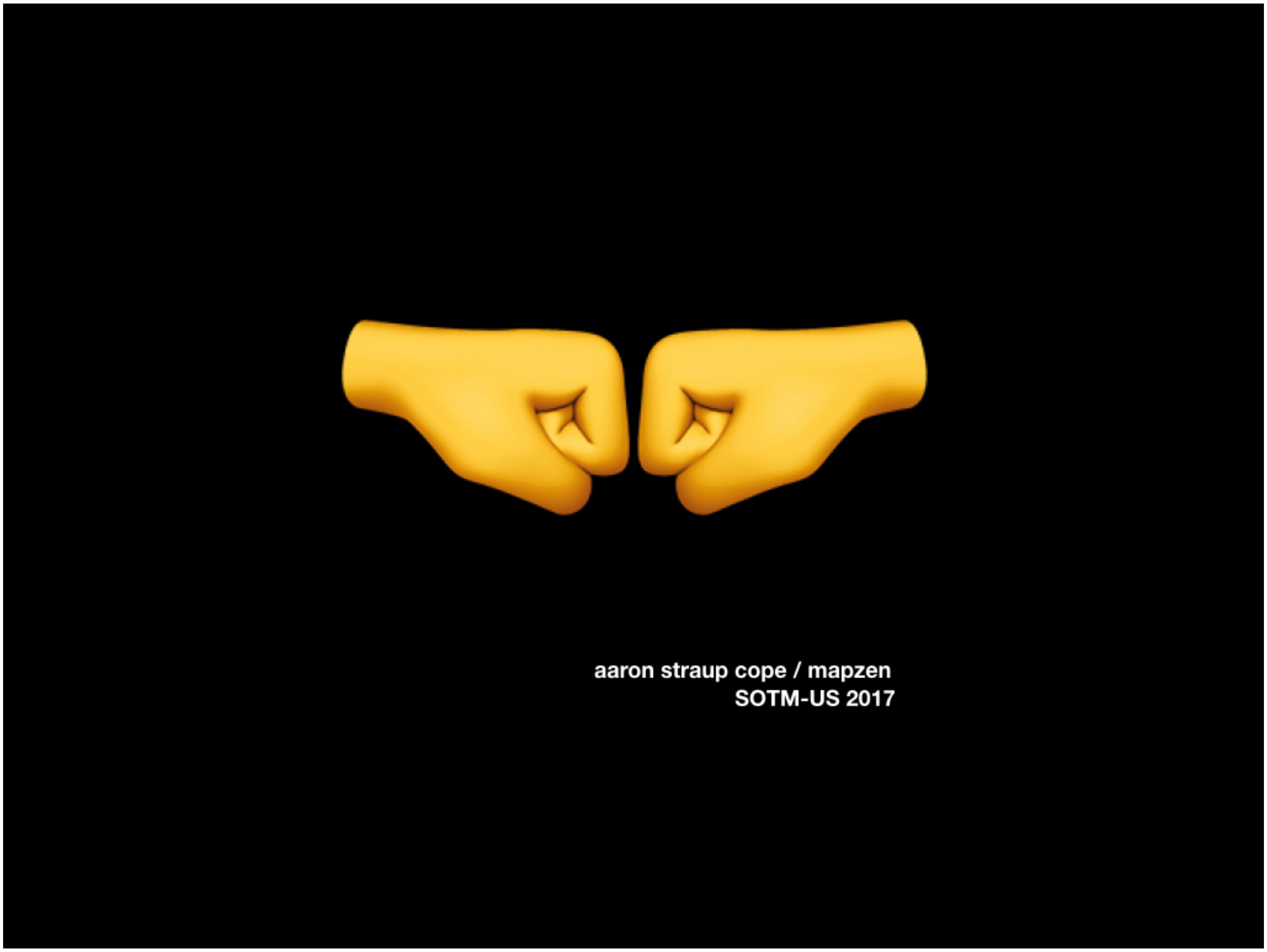
David Nesbitt

Dave leads Mapzen Mobility engineering. Rides a variety of 2 wheel vehicles.

© 2017 Mapzen

Who's On First :fist-bump: OpenStreetMap

osm (/tag/osm) whosonfirst (/tag/whosonfirst)



aaron straup cope / mapzen
SOTM-US 2017

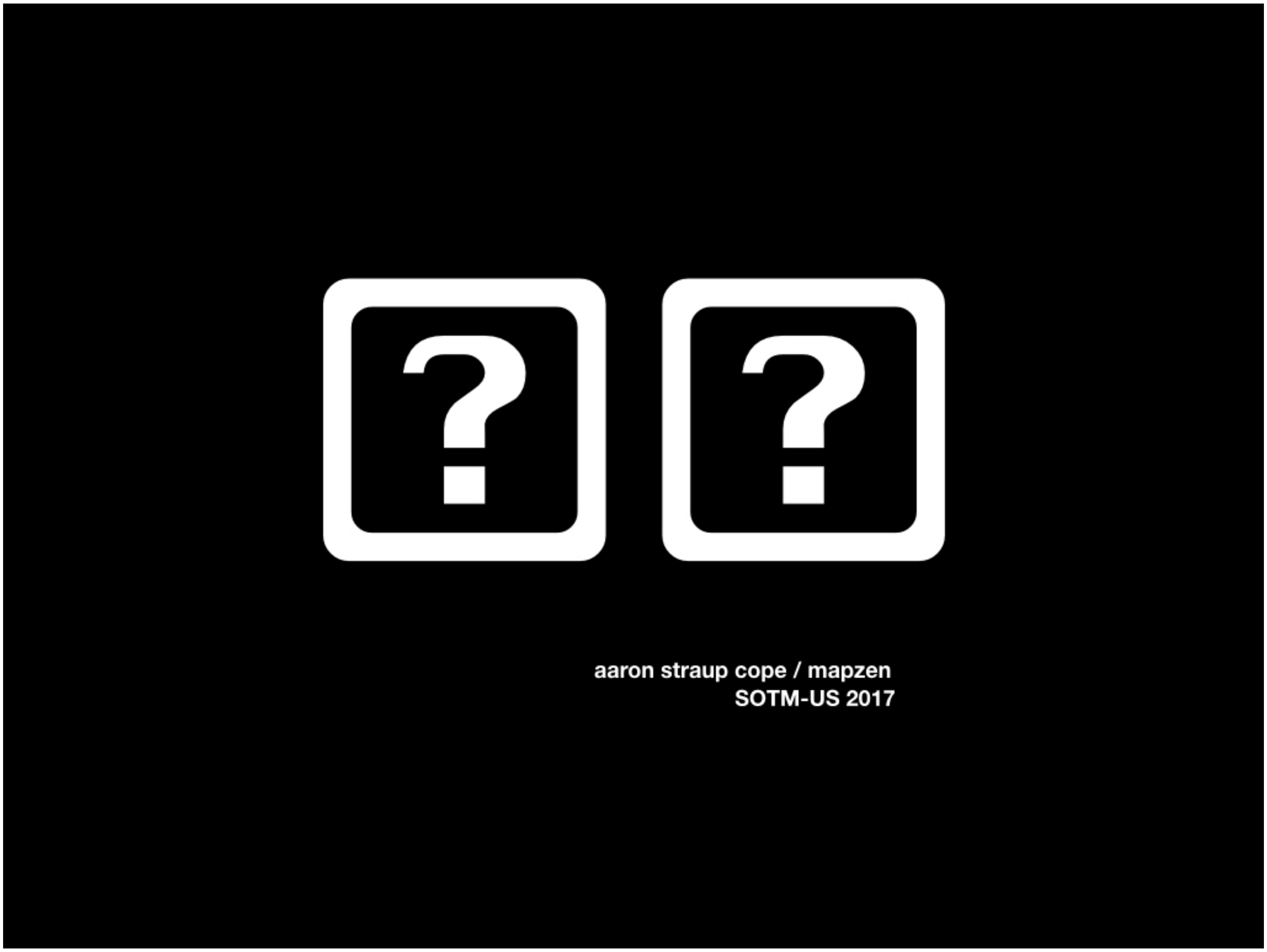
(https://mapzen.com/blog/whosonfirst-sotmus-2017#intro)

Hello. I haven't been able to attend **State of the Map (https://2017.stateofthemap.us)** since **2014 (https://2014.stateofthemap.us)** so it's good to be back. Thank you for inviting me to speak.

Before we get started I'd like to do a quick bit of house-keeping.

The full title of this talk is “Who’s On First colon fist-bump colon OpenStreetMap". The “colon fist-bump colon” part is meant to be a mock **emoji shortcode (http://www.webstock.org.nz/talks/emoji-fun-profit/)**.

There is not an emoji for **“fist-bump” (https://www.flickr.com/photos/secret_canadian/galleries/72157623155364950/)** yet but I’m hoping there will be soon. Now that the **LEFT-FACING FIST (https://emojipedia.org/left-facing-fist/)** and **RIGHT-FACING FIST (https://emojipedia.org/right-facing-fist/)** characters have been added to Unicode 10 all that is necessary to **create a FIST-BUMP emoji (https://unicode.org/emoji/selection.html)** is a combining character.

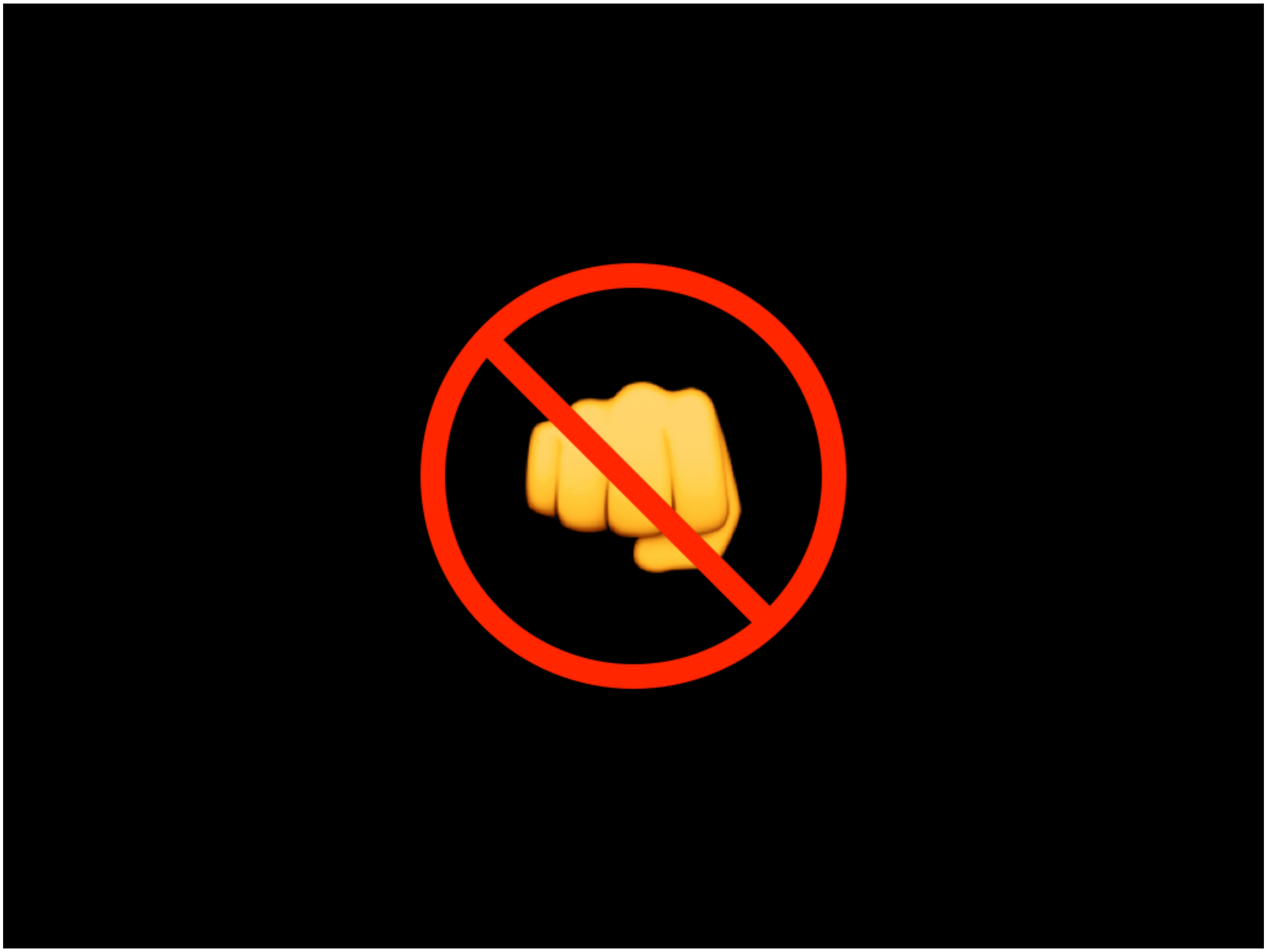

aaron straup cope / mapzen
SOTM-US 2017

(https://mapzen.com/blog/whosonfirst-sotmus-2017#unicode)

Not many operating systems have adopted **Unicode 10 (https://www.unicode.org/versions/Unicode10.0.0/)** yet, which is why I included a fake emoji shortcode.

For example, that last slide is actually an image. I prepared this talk on a different computer and the laptop I brought to the conference still **can't display Unicode 10 characters (https://emojipedia.org/unicode-10.0/)**.

Tangentially related the software used to publish this blog gets upset when you use the standard ***COLON (https://thisisaaronland.github.io/unicode-table/#58)*** *character in a title which is why we've used the less common* ***MODIFIER LETTER COLON (https://thisisaaronland.github.io/unicode-table/#42889)*** *character in it's place…*



(https://mapzen.com/blog/whosonfirst-sotmus-2017#punching)

Somewhere between submitting this talk for consideration and its inclusion in the conference program the title was lost in translation.

Instead of newly minted Unicode characters (which would almost certainly **not render (https://www.flickr.com/photos/bees/9458878023)** on a majority of people's screens, never mind the computers used by the printers who made the conference schedule) or my fake emoji shortcode the decision was made to use the more readily available **ONCOMING FIST (https://emojipedia.org/fisted-hand-sign/)** character.

I am not really bothered by this except to point out that the title takes on a slightly more threatening and hostile tone than I ever intended.

For the record, punching **OpenSteetMap (https://openstreetmap.org)** (OSM) is not what I had in mind.

(https://mapzen.com/blog/whosonfirst-sotmus-2017#wondertwins)

This is more what I had in mind.

For anyone who doesn't understand what's going on here, it's two people dressed up as **the Wonder Twins (https://en.wikipedia.org/wiki/Wonder_Twins)**.

The Wonder Twins were Saturday morning cartoon characters, part of the Super Friends alongside more familiar characters like Batman and Wonder Woman.

The Wonder Twins would "activate" their super powers by fist-bumping upon which the siblings would transform in to a pair of complimentary objects. Perhaps **the most famous, and ridiculous, pairing (https://www.youtube.com/watch?v=FouDjI3GzEI)** saw one of the twins become a giant eagle which carried the other who had transformed in to a bucket of water.

The 70s were weird like that in a way that we don't have time to discuss today except to say that Who's On First would like to be the bucket of water to OpenStreetMap's giant eagle.

whosonfirst.mapzen.com

@allofthethings

@thisisaaronland

(https://mapzen.com/blog/whosonfirst-sotmus-2017#whosonfirst)

Who's On First (https://whosonfirst.mapzen.com) (WOF) is a gazetteer. If you've never heard the term "gazetteer" before it's basically just a big phone book, but a phone book of places rather than people. I am going to spend most of this talk discussing venues but I want to start with a high-level overview of what Who's On First is. The shape of the elephant so to speak.

- It is an **openly licensed (https://whosonfirst.mapzen.com/docs/licenses/)** dataset. At its most restrictive, data is published under a Creative Commons By-Attribution license. Whenever possible, and this is true of our own day-to-day work, data is published under a

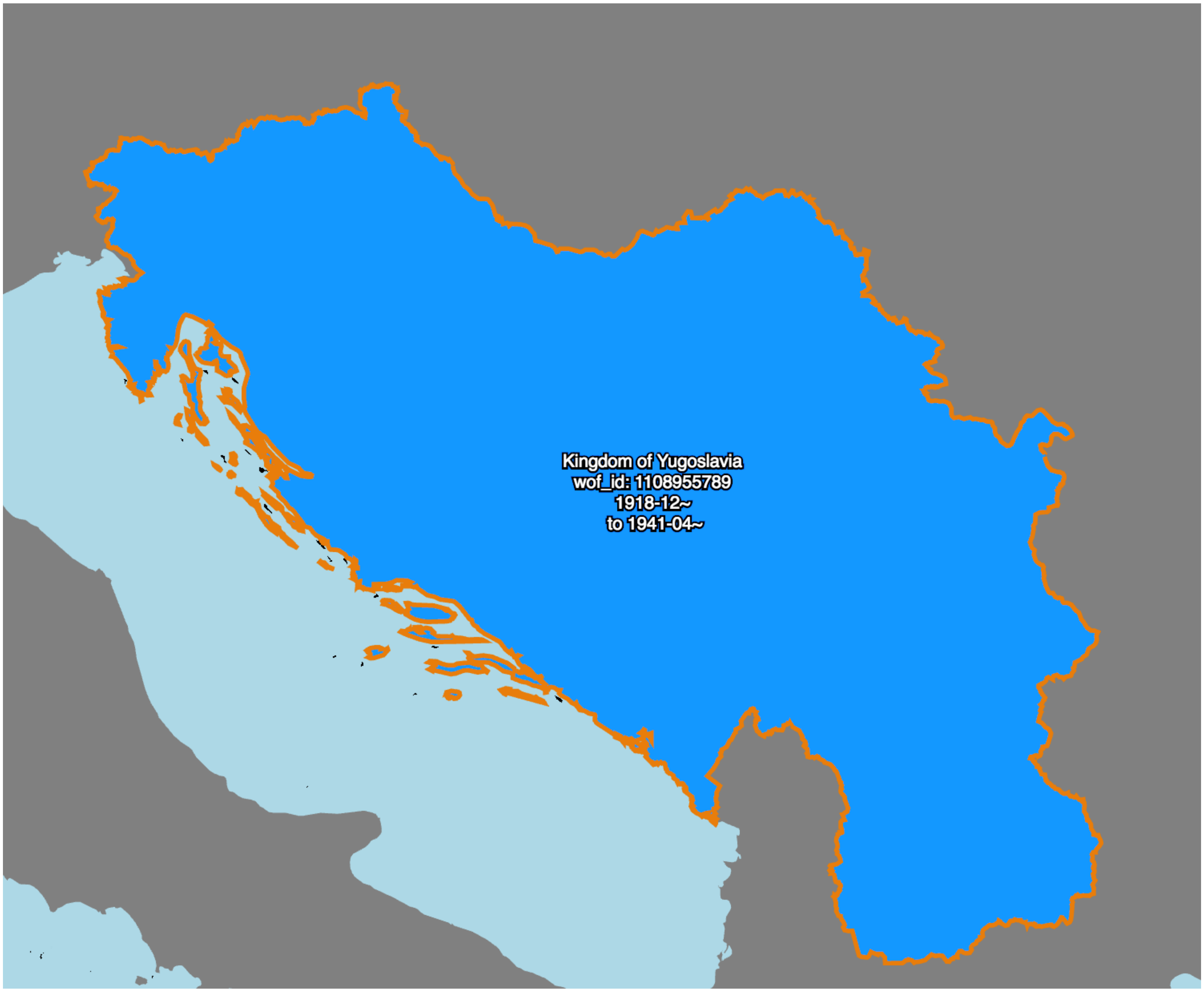
Creative Commons Zero public domain license.

- Every record in Who's On First has **a stable permanent and unique numeric identifier (https://whosonfirst.mapzen.com/data/principles/)**. There are no semantics encoded in the IDs.
- At rest, each record is stored as a plain-text **GeoJSON (https://tools.ietf.org/html/rfc7946)** file. Our goal is to ensure that Who's On First embodies the principals of portability, durability and longevity. This led us to adopt plain-vanilla text files as the base unit of delivery.
- Files are stored in a nested hierarchy of directories derived from their IDs.
- There are **a common set of properties (https://whosonfirst.mapzen.com/docs/properties/)** applied to all records which may be supplemented by an arbitrary number of additional properties specific to that place.
- There are **a finite number of place types (https://github.com/whosonfirst/whosonfirst-placetypes#here-is-a-pretty-picture)** in Who's On First and all records share a common set of ancestors. As with properties, any given record may have **as complex a hierarchy as the circumstances demand (https://whosonfirst.mapzen.com/docs/hierarchies/)** but there is a shared baseline hierarchy across the entire dataset.
- Individual records may have multiple geometries or multiple hierarchies and sometimes both.
- Records may be updated or superseded, cessated or even deprecated. Once a record is created though it can never be removed or replaced.
- Lastly and most importantly Who's On First is meant, by design, to accommodate all of the places.

Who's On First is not a carefully selected list of important or relevant places. It is not meant to act as the threshold by which places are measured. Who's On First, instead, is meant to provide the raw material with which a multiplicity of thresholds might be created.

From Who's On First's perspective **the point is not that one place is more important or relevant than any other (https://mapzen.com/blog/who-s-on-first/)**. The point is not that a place may or may not exist anymore or that its legitimacy as a place is disputed. The point is, critically, that people *believe* them to exist or to have existed.

This is why I often refer to Who's On First as **"a gazetteer of consensual hallucinations" (https://mapzen.com/blog/mapping-with-bias/)**.



(https://mapzen.com/blog/whosonfirst-sotmus-2017#yugoslavia)

Here's an example.

This is an animation of **work that my colleague Stephen Epps did (https://mapzen.com/blog/tackling-space-and-time-in-whosonfirst/)** earlier this year to create and link all the records representing the many ways that **the place we used to call "Yugoslavia" (https://whosonfirst.mapzen.com/spelunker/search/?q=yugoslavia)** has changed between the start of the 20th century and now.

I'm going to let this run on a loop while I discuss a point some of you may have tweaked to in the last slide. Specifically that OpenStreetMap and Who's On First have sufficiently different licensing requirements that they prevent data exchange in both directions.

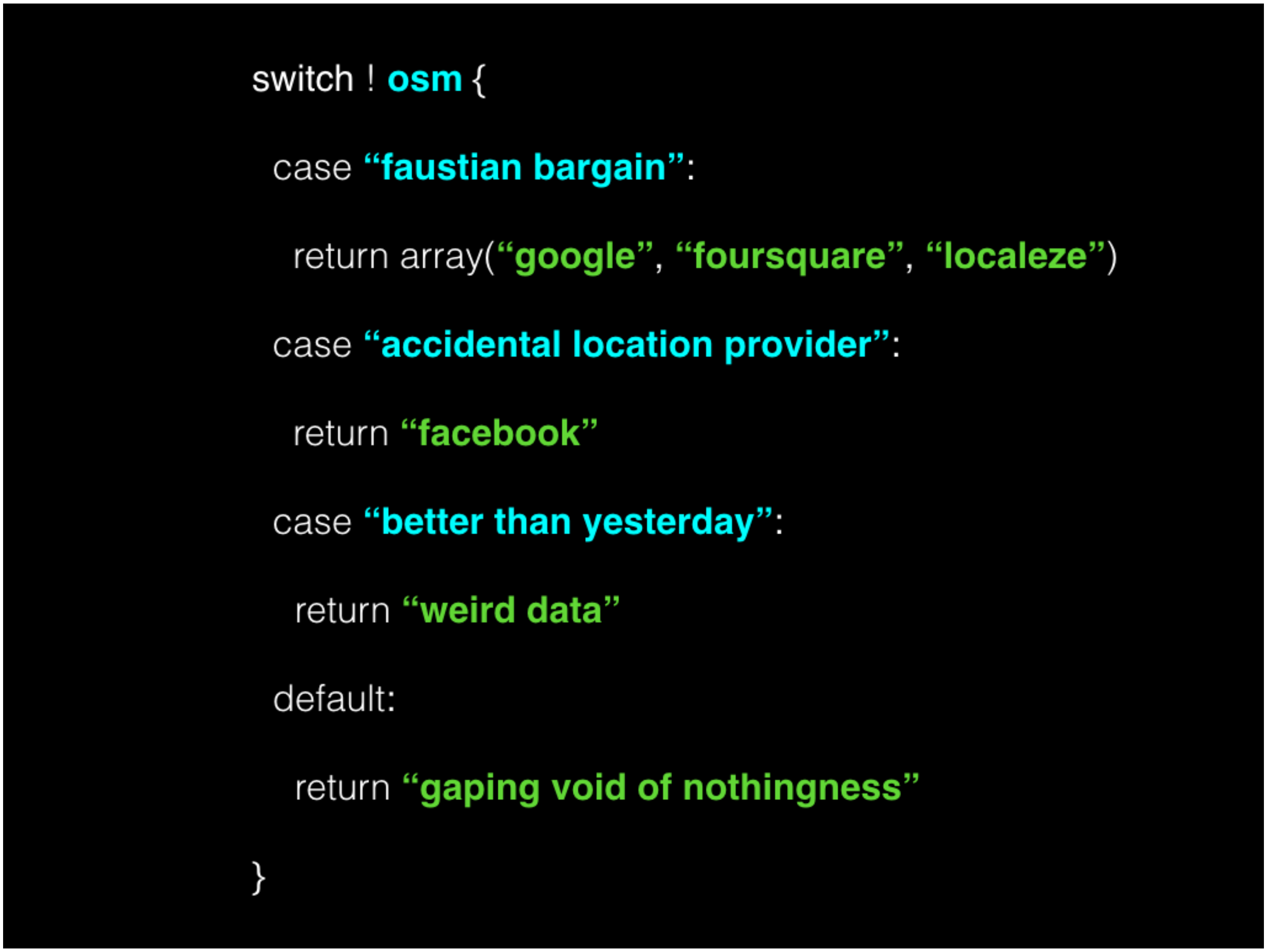
I just said “license” which means somewhere, someone just won **“OSM Bingo” (https://wiki.openstreetmap.org/wiki/Proposed_features/bingo_hall)**.

Seriously though I’d like to make one thing perfectly clear before I go any further: I am not here, either as an individual or an employee of Mapzen, to suggest *in any way* that the OpenStreetMap community revisit **the decision to adopt the ODbL (https://wiki.openstreetmap.org/wiki/Open_Database_License)**.

My own personal feeling is that by virtue of having accomplished the impossible in creating OpenStreetMap, and having done so in thirteen short years, that the community has earned the right to do whatever it wants.

At the same time I still need and want an openly licensed location database that can be used and adapted in both commercial and “closed” projects without any restrictions beyond attribution.

This is the space that Who’s On First occupies and it is why we will not – can not – import data from OpenStreetMap.



```
switch ! osm {
  case “faustian bargain”:
    return array(“google”, “foursquare”, “localeze”)
  case “accidental location provider”:
    return “facebook”
  case “better than yesterday”:
    return “weird data”
  default:
    return “gaping void of nothingness”
}
```

(https://mapzen.com/blog/whosonfirst-sotmus-2017#ifnotosm)

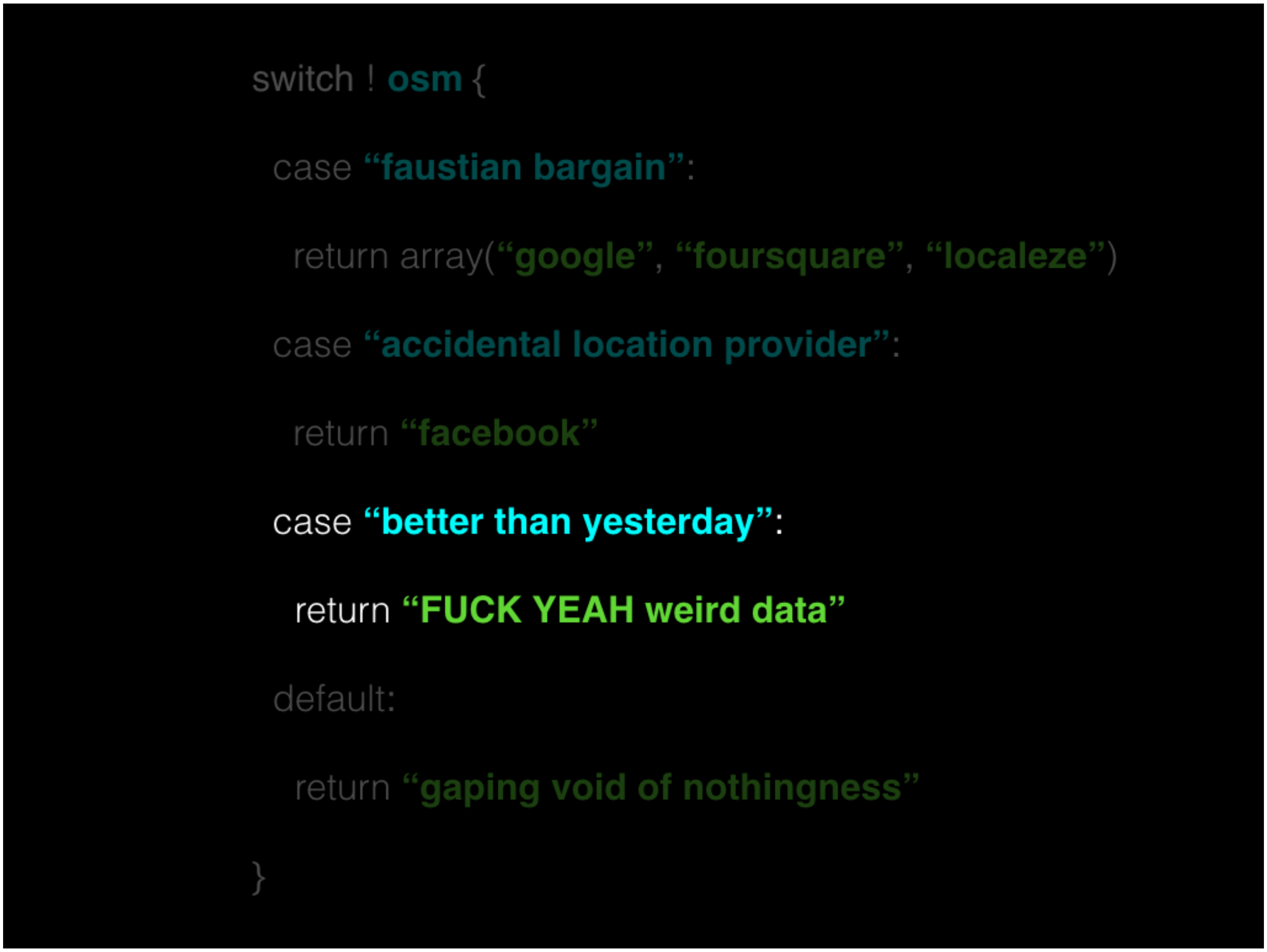
About this time, two years ago, I had a little freak out at work.

We had been discussing the availability and viability of open venue datasets. The reason I freaked out is that when you take OpenStreetMap off the table, for the reasons I’ve just outlined, there are effectively *no openly licensed venue databases* so I wasn’t sure what we were talking about.

Accurate and up-to-date venue data is a difficult and daunting challenge. There are a few companies that have built successful businesses collecting and reselling that data under terms which can only be described as a Faustian bargain.

Other companies like Facebook, we’re told, have amassed similar and sometimes superior catalogs of data almost by accident as a by-product of their day-to-day work. So far, though, none of these companies has seen fit to share any of that data.

Which really only leaves two alternatives. The first is the gaping void of nothingness that we've all come to know and love. The second is to embrace the idea that *something is better than nothing* and that something which can be improved upon over time is even better than that.



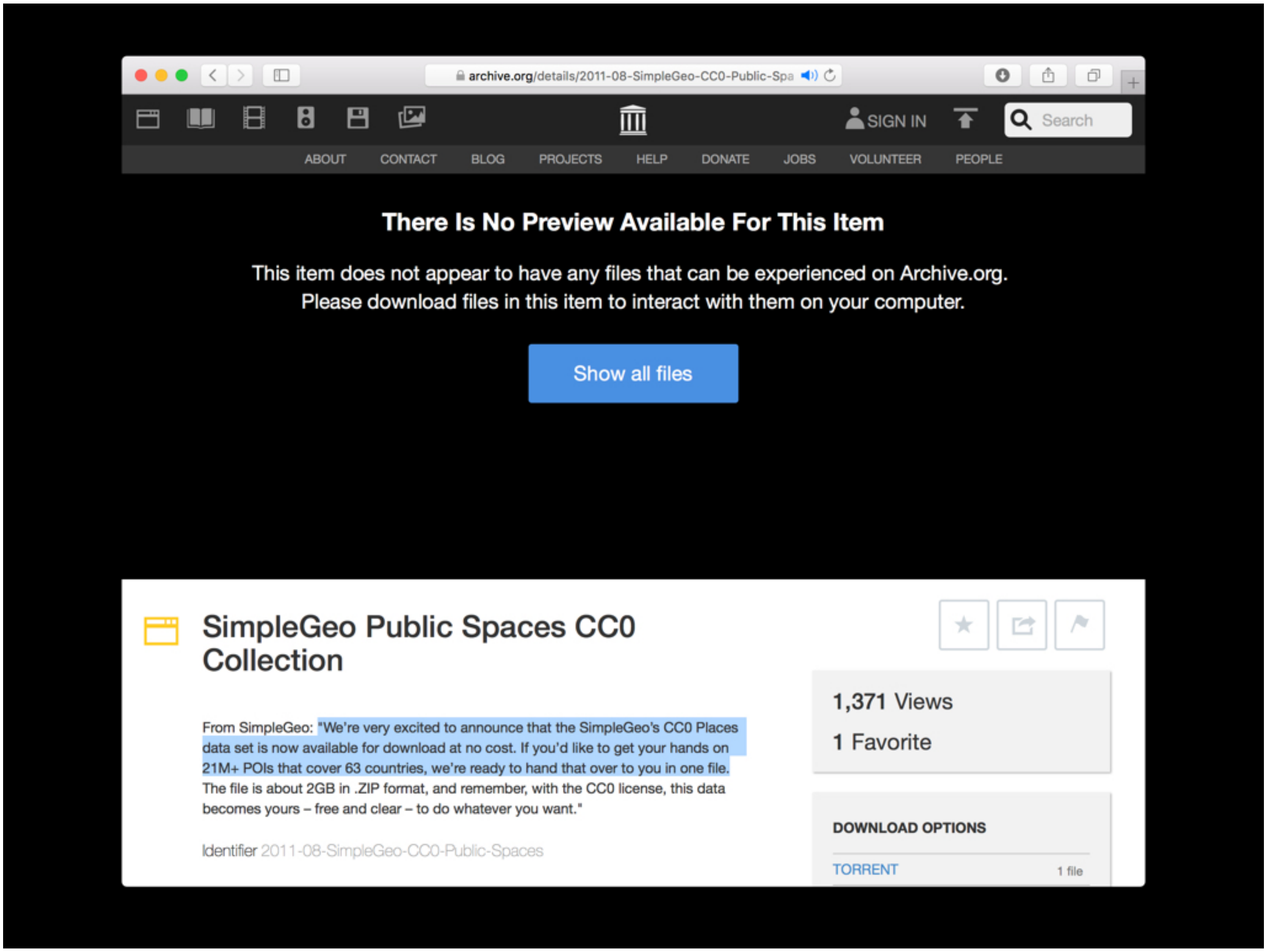
```
switch ! osm {
  case "faustian bargain":
    return array("google", "foursquare", "localeze")
  case "accidental location provider":
    return "facebook"
  case "better than yesterday":
    return "FUCK YEAH weird data"
  default:
    return "gaping void of nothingness"
}
```

(https://mapzen.com/blog/whosonfirst-sotmus-2017#absence)

To embrace, or at least accept, that our burden in the near-term will be one of managing absence.

That *in addition* to all the hard work of collecting, vetting and improving data we will also need to think about ways to talk to people about the data we don't have and to develop interfaces for buffering and tempering that absence.

This makes an already daunting project exhausting even to think about.


archive.org/details/2011-08-SimpleGeo-CC0-Public-Spa
SIGN IN
Search
ABOUT
CONTACT
BLOG
PROJECTS
HELP
DONATE
JOBS
VOLUNTEER
PEOPLE
Show all files
1,371 Views
1 Favorite
DOWNLOAD OPTIONS
TORRENT
1 file

There Is No Preview Available For This Item

This item does not appear to have any files that can be experienced on Archive.org. Please download files in this item to interact with them on your computer.

SimpleGeo Public Spaces CC0 Collection

From SimpleGeo: "We're very excited to announce that the SimpleGeo's CC0 Places data set is now available for download at no cost. If you'd like to get your hands on 21M+ POIs that cover 63 countries, we're ready to hand that over to you in one file. The file is about 2GB in .ZIP format, and remember, with the CC0 license, this data becomes yours – free and clear – to do whatever you want."

Identifier 2011-08-SimpleGeo-CC0-Public-Spaces

(https://mapzen.com/blog/whosonfirst-sotmus-2017#simplegeo)

The good news is that there was in fact one open venue database, available for use, in 2015. In 2011 the geo-services company SimpleGeo **published their Places database (https://web.archive.org/web/20110514083813/http://blog.simplegeo.com/2011/04/20/open-places-data/)**, containing 21 million business listings, under a Creative Commons Zero public domain license.

The company went out of business shortly after that but not before **Jason Scott (https://t.co/1f0N0IPkEt)** grabbed a copy of the data and **put it on the Internet Archive (https://archive.org/details/2011-08-SimpleGeo-CC0-Public-Spaces)**.

The first thing to know about the SimpleGeo data is that it is a flawed dataset.

8 years “out of date”

(https://mapzen.com/blog/whosonfirst-sotmus-2017#outofdate)

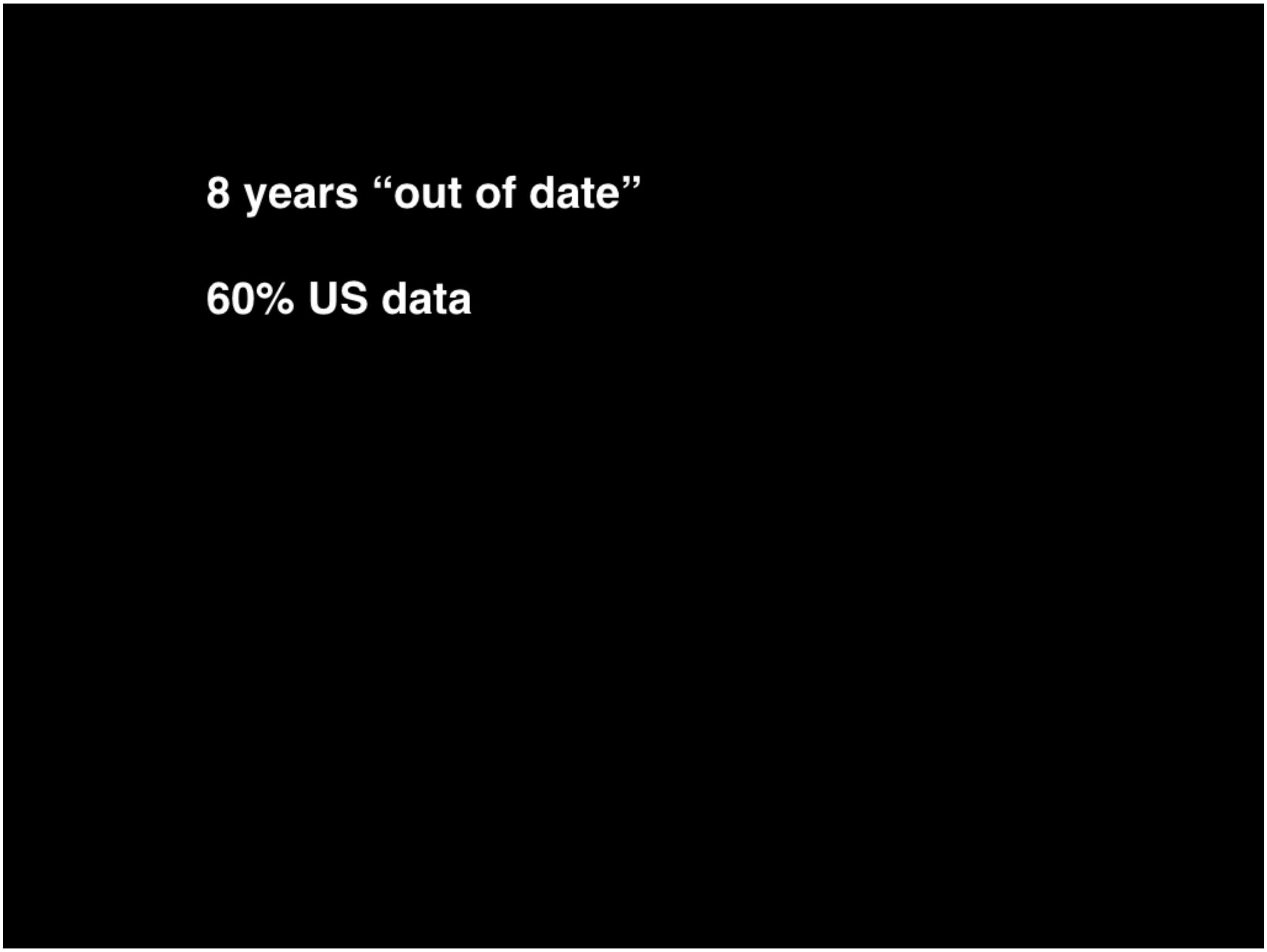
For starters it is almost 8 years “out of date”.

That’s what we say but when we say “out of date” what we are often really saying is there is no *new* stuff.

And when we say “no new stuff” what we’re often really saying is that there is no new stuff for a *particular kind* of business, typically those focused on snacking and grooming and nesting and aimed at a particular demographic: 18-35 year olds, with disposable income.

The funny thing is that lots of businesses survive longer than 17 years. Think of **the butcher that’s been in your town (http://places.mapzen.com/id/572151977)** for 75 or 80 years now. If we’re lucky, sooner or later we all get older and eventually need to call a plumber or an electrician or a notary.

That’s the stuff the SimpleGeo dataset is made of so to say that it’s out of date seems inaccurate and unfair.

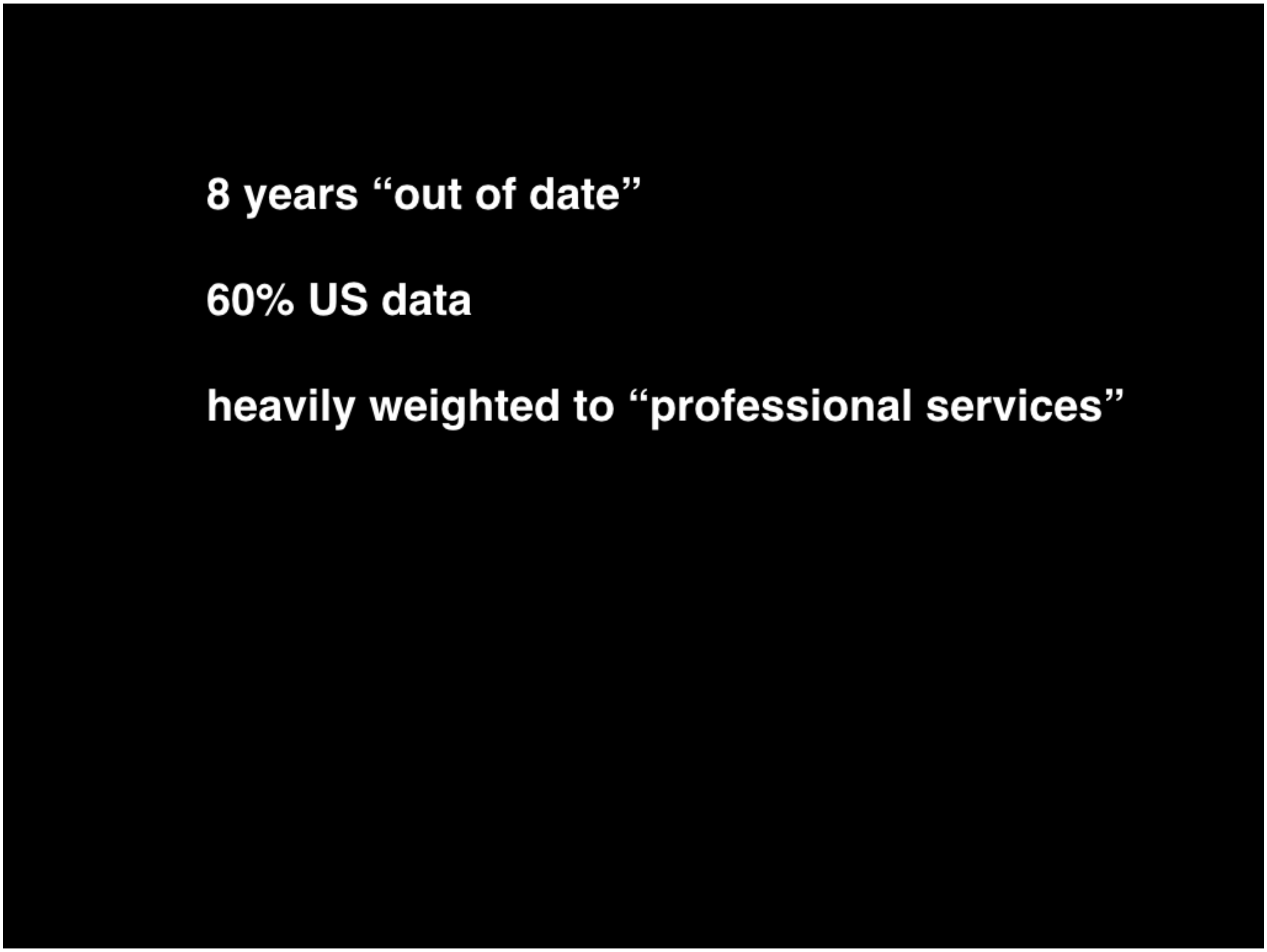


8 years “out of date”

60% US data

(https://mapzen.com/blog/whosonfirst-sotmus-2017#usa)

The data is **heavily slanted towards the US (https://whosonfirst.mapzen.com/spelunker/placetypes/venue/)**. SimpleGeo claimed the data covered 60 countries but that is disingenuous when you consider all but about a dozen countries have fewer than a hundred venues.



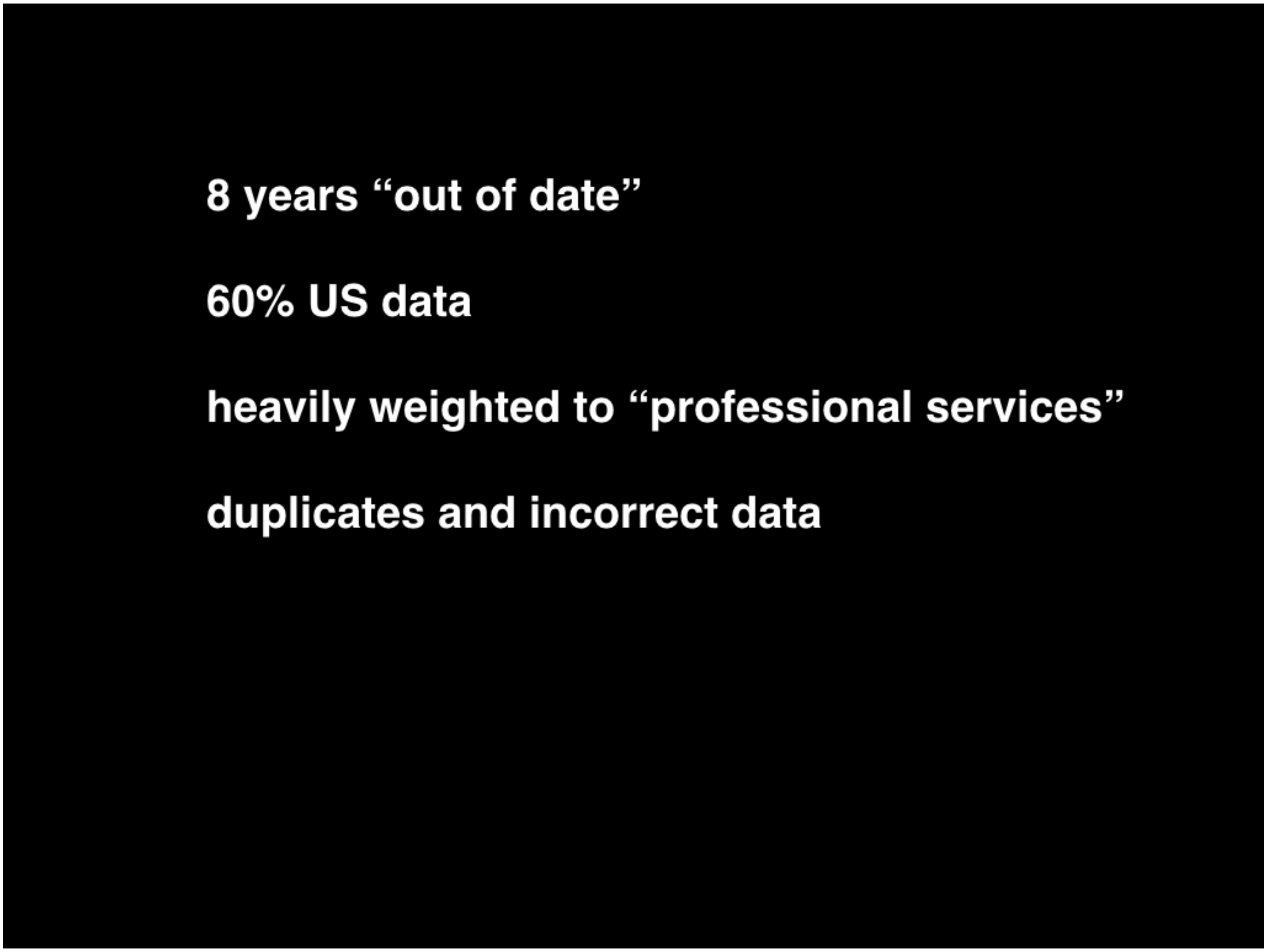
8 years “out of date”

60% US data

heavily weighted to “professional services”

(https://mapzen.com/blog/whosonfirst-sotmus-2017#service)

The data is also heavily weighted towards professional services. There are a lot of CPAs and lawyers and doctors and, to be honest, the sheer volume can make it difficult to see the trees for the forest.



8 years “out of date”

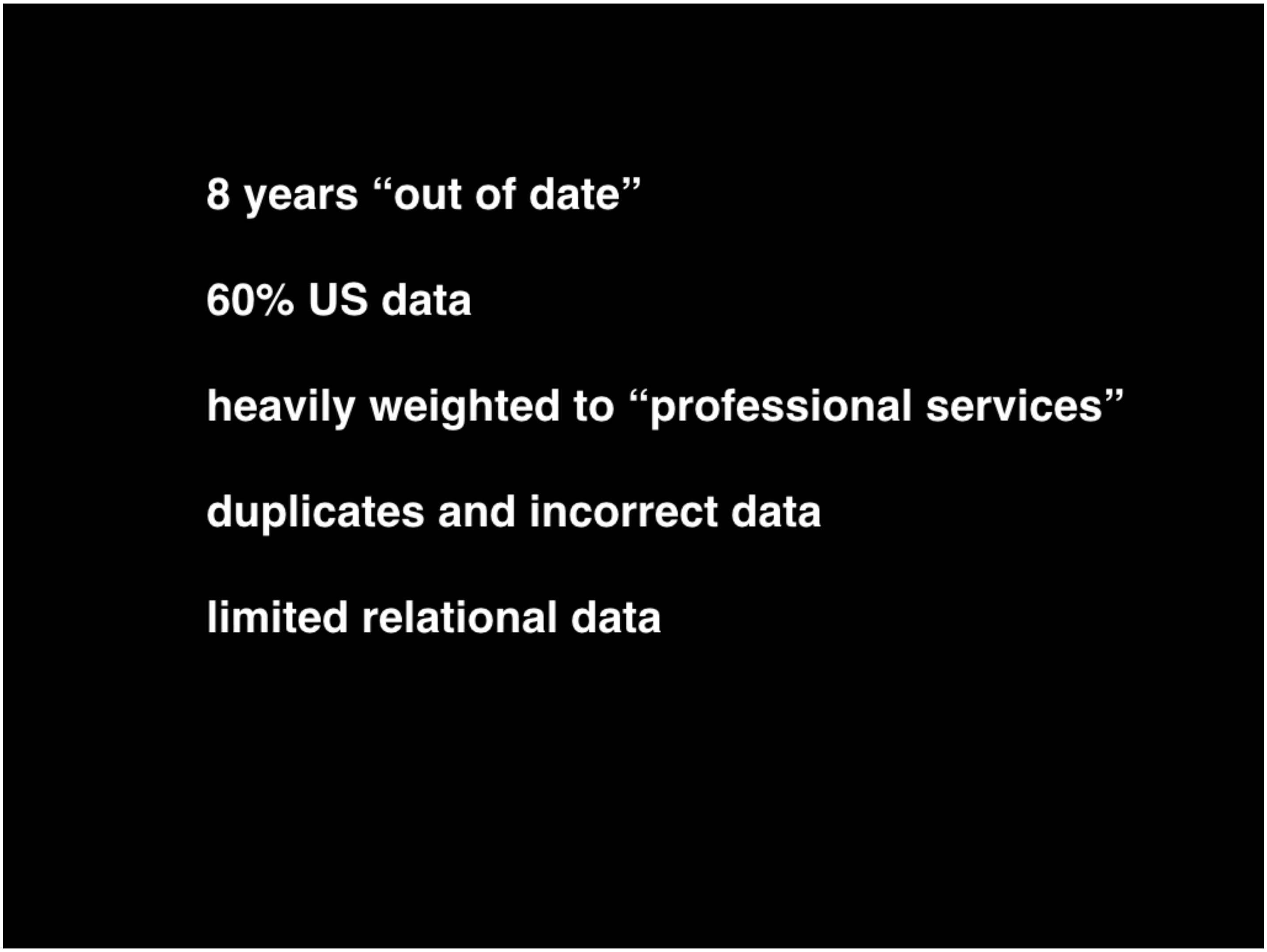
60% US data

heavily weighted to “professional services”

duplicates and incorrect data

(https://mapzen.com/blog/whosonfirst-sotmus-2017#data)

There is some genuinely bad data. No surprises there and I doubt anyone here is ready to cast the first stone when it comes to bad data.



8 years “out of date”

60% US data

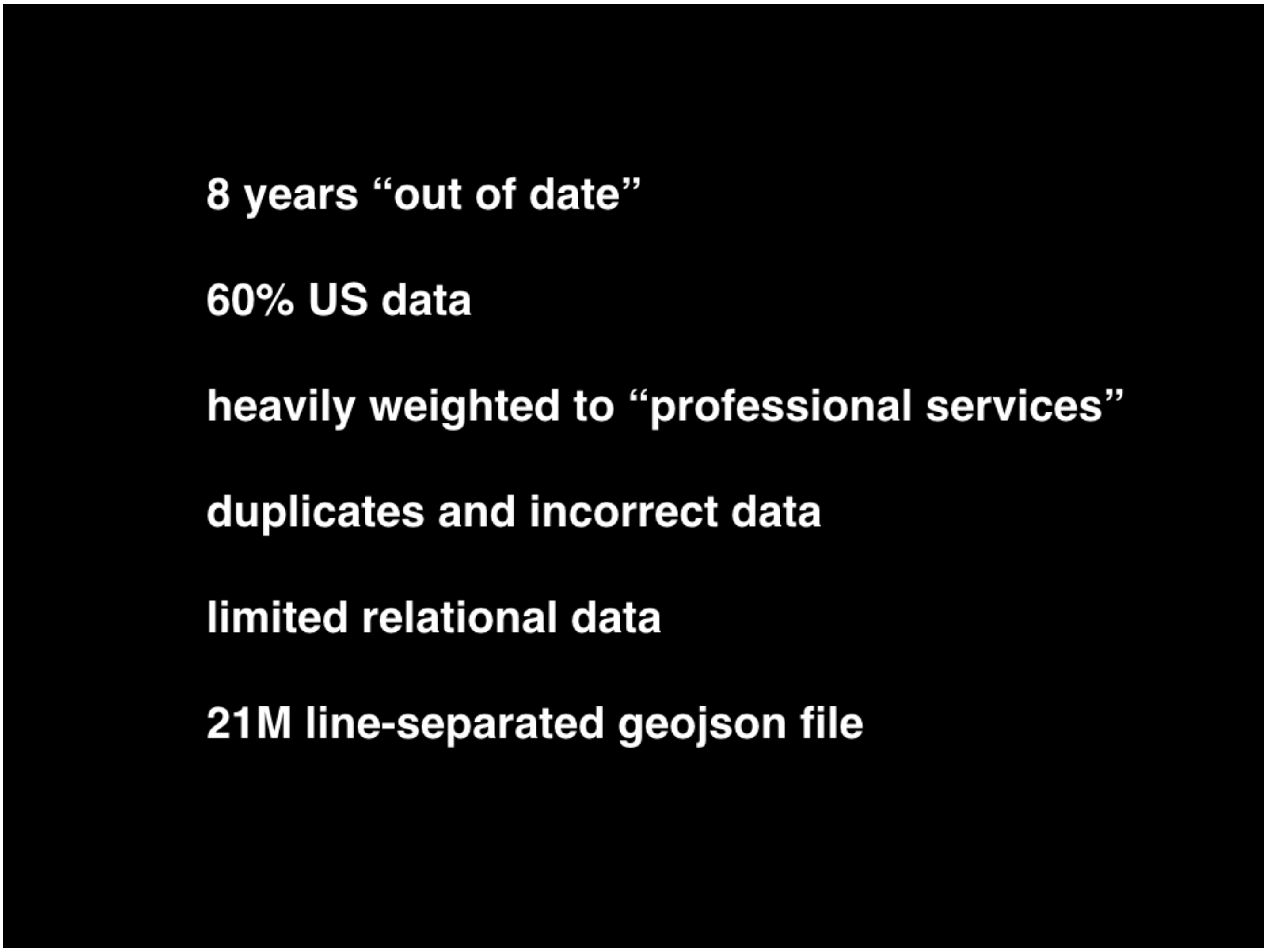
heavily weighted to “professional services”

duplicates and incorrect data

limited relational data

(https://mapzen.com/blog/whosonfirst-sotmus-2017#relational)

There is minimal structured relational data. For example, the only way to find all the records in a given a locality or neighbourhood is to load everything in to a database and perform spatial queries.



8 years “out of date”

60% US data

heavily weighted to “professional services”

duplicates and incorrect data

limited relational data

21M line-separated geojson file

(https://mapzen.com/blog/whosonfirst-sotmus-2017#geojson)

Everything was published as a single line-separated GeoJSON file. Had I been in SimpleGeo’s shoes at the time I might have made the same decision. As a consumer of that data, the decision has been nothing but 100% sad-making.

better than yesterday

(https://mapzen.com/blog/whosonfirst-sotmus-2017#better)

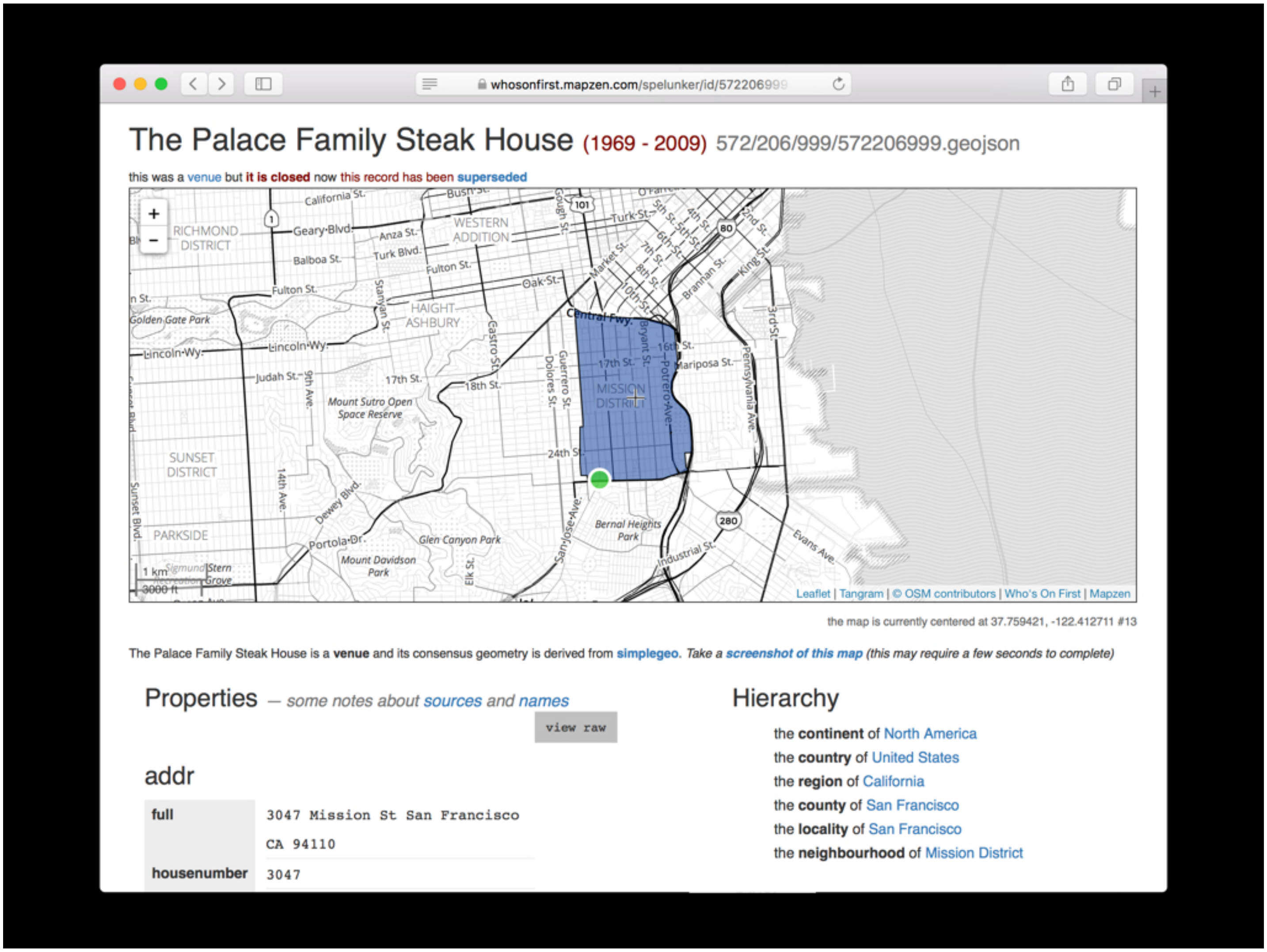
But it’s a good start and the first thing we did was to **explode that 21 million line text file in to 21 million discrete records (https://mapzen.com/blog/whosonfirst-venues)**, each with a new Who’s On First ID.

If nothing else the data has proven its value in forcing us to think about how, from a purely operational point of view, we manage editing, storing and distributing that much data.

Without getting in to all the details I’ll just say that **trying to put 21 million files in to a single Git repository (https://github.com/whosonfirst-data)** remains challenging enough to the point of being impossible.

Simply having 21 million files on a single volume, in 2017, still causes people to **run out of inodes (https://www.ibm.com/developerworks/aix/library/au-speakingunix14/index.html)** and try explaining that to someone who doesn’t understand, or care, how filesystems work.

But there are more than 21 million venues in the world and we're going to have to figure these things out eventually so we might as well start now.


whosonfirst.mapzen.com/spelunker/id/572206999
RICHMOND DISTRICT
WESTERN ADDITION
HAIGHT ASHBURY
Golden Gate Park
MISSION DISTRICT
SUNSET DISTRICT
PARKSIDE
Mount Sutro Open Space Reserve
Glen Canyon Park
Mount Davidson Park
Bernal Heights Park
Central Fwy.
California St.
Geary Blvd.
Fulton St.
Lincoln Wy.
Judah St.
Oak St.
17th St.
18th St.
24th St.
16th St.
Mariposa St.
Market St.
Brannan St.
King St.
3rd St.
Evans Ave.
Industrial St.
Portola Dr.
Dewey Blvd.
San Jose Ave.
Castro St.
Dolores St.
Guerrero St.
Bryant St.
Potrero Ave.
Pennsylvania Ave.
Sunset Blvd.
9th Ave.
14th Ave.
Stanyan St.
101
80
280
1 km
3000 ft
Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen
view raw

The Palace Family Steak House (1969 - 2009) 572/206/999/572206999.geojson

this was a venue but **it is closed** now this record has been **superseded**

the map is currently centered at 37.759421, -122.412711 #13

The Palace Family Steak House is a **venue** and its consensus geometry is derived from **simplegeo**. *Take a* ***screenshot of this map*** *(this may require a few seconds to complete)*

Properties — *some notes about sources and names*

addr

full	3047 Mission St San Francisco CA 94110
housenumber	3047

Hierarchy

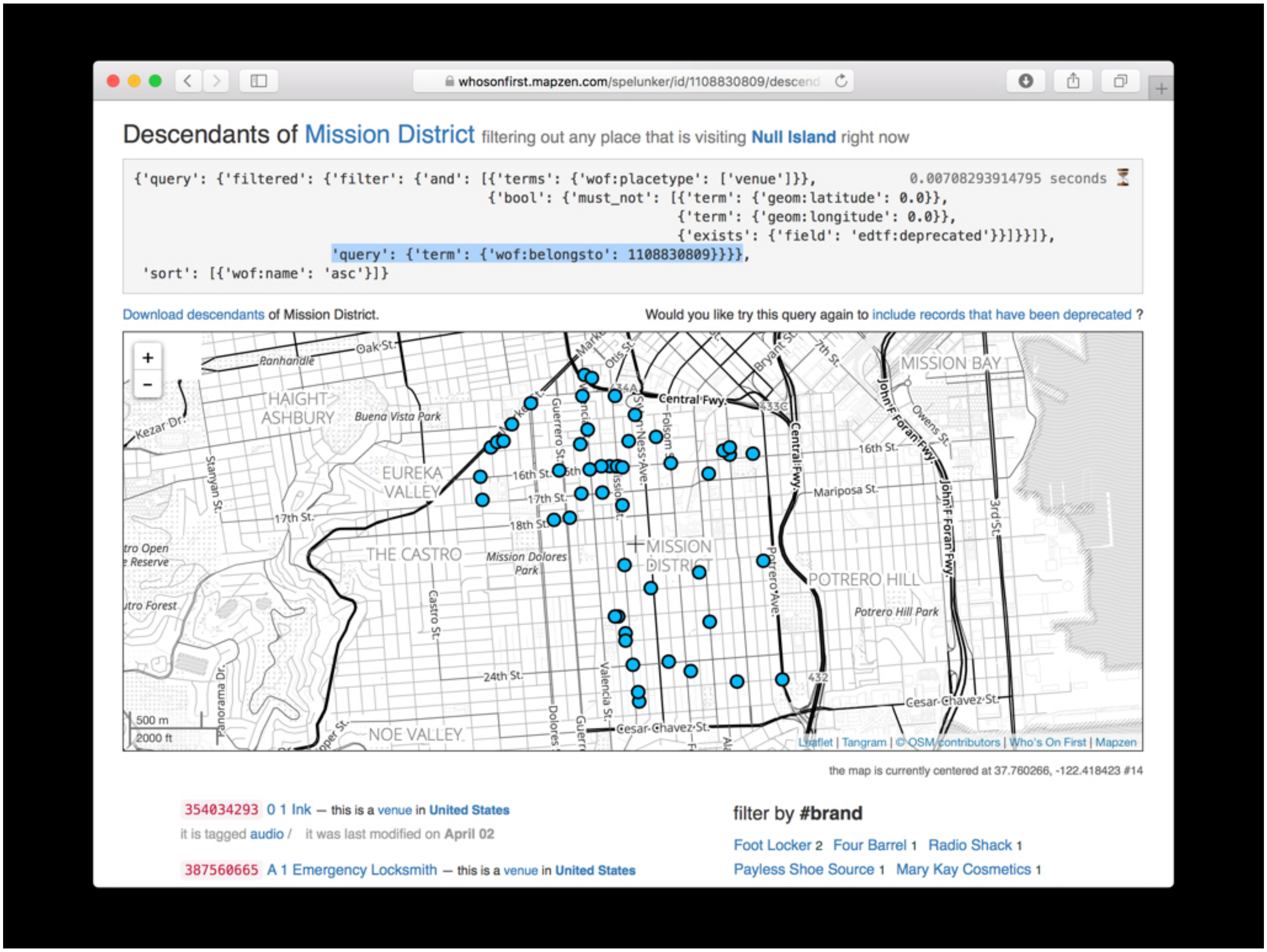
the **continent** of North America
the **country** of United States
the **region** of California
the **county** of San Francisco
the **locality** of San Francisco
the **neighbourhood** of Mission District

(https://mapzen.com/blog/whosonfirst-sotmus-2017#palace)

This is what that looks like and if all the venues in the world weren't enough, we also want to be able to track historical venues.

Let's pretend that Who's On First existed in 1985 and a writer has published a review of **The Palace Steak House (http://places.mapzen.com/id/572206999)**, associating its Who's On First ID with their article.

It is important that both the author and the readers of that article can rely on that ID with the confidence that its meaning won't change even though the Palace went out of business in 2009 and that space has been occupied by **three (https://places.mapzen.com/id/1108800487) different (https://places.mapzen.com/id/1108800489) restaurants (https://places.mapzen.com/id/1126130035)** since then.


whosonfirst.mapzen.com/spelunker/id/1108830809/descend
+
−
Panhandle
Oak St.
HAIGHT ASHBURY
Buena Vista Park
Kezar Dr.
Stanyan St.
EUREKA VALLEY
17th St.
16th St.
18th St.
THE CASTRO
Mission Dolores Park
Castro St.
Guerrero St.
Valencia St.
Dolores St.
South Van Ness Ave.
Folsom St.
Central Fwy.
MISSION DISTRICT
MISSION BAY
16th St.
Mariposa St.
Potrero Ave.
POTRERO HILL
Potrero Hill Park
John F Foran Fwy.
3rd St.
Owens St.
Bryant St.
7th St.
24th St.
Cesar Chavez St.
NOE VALLEY
Panorama Dr.
500 m
2000 ft
Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen

Descendants of Mission District filtering out any place that is visiting Null Island right now

```
{'query': {'filtered': {'filter': {'and': [{'terms': {'wof:placetype': ['venue']}},          0.00708293914795 seconds
                                            {'bool': {'must_not': [{'term': {'geom:latitude': 0.0}},
                                                                   {'term': {'geom:longitude': 0.0}},
                                                                   {'exists': {'field': 'edtf:deprecated'}}]}}]},
                        'query': {'term': {'wof:belongsto': 1108830809}}}},
 'sort': [{'wof:name': 'asc'}]}
```

Download descendants of Mission District.

Would you like try this query again to include records that have been deprecated ?

the map is currently centered at 37.760266, -122.418423 #14

354034293 0 1 Ink — this is a venue in United States

it is tagged audio / it was last modified on April 02

387560665 A 1 Emergency Locksmith — this is a venue in United States

filter by **#brand**

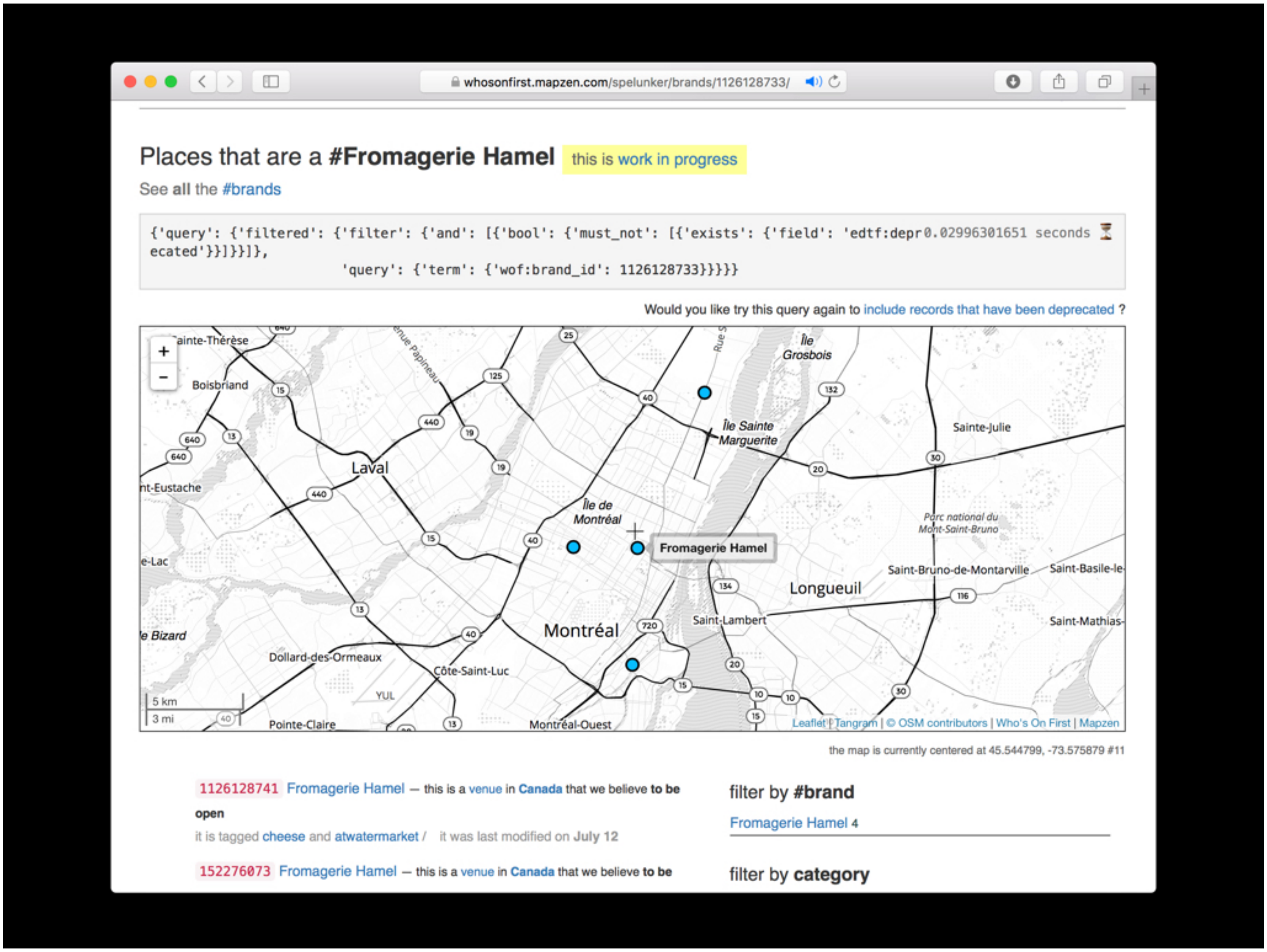
Foot Locker 2 Four Barrel 1 Radio Shack 1

Payless Shoe Source 1 Mary Kay Cosmetics 1

(https://mapzen.com/blog/whosonfirst-sotmus-2017#mission)

The second thing we did was assign each venue a hierarchy with pointers to all the other places in Who’s On First that venue holds hands with.

If nothing else this reduces the problem of **finding all the venues in the Mission (https://whosonfirst.mapzen.com/spelunker/id/1108830809/descendants/?exclude=nullisland&placetype=venue)** to a simple two-part key/value query.


whosonfirst.mapzen.com/spelunker/brands/1126128733/
Sainte-Thérèse
Boisbriand
Laval
Île Grosbois
Île Sainte Marguerite
Sainte-Julie
Île de Montréal
Fromagerie Hamel
Parc national du Mont-Saint-Bruno
Saint-Bruno-de-Montarville
Saint-Basile-le-
Longueuil
Saint-Lambert
Saint-Mathias-
Montréal
Dollard-des-Ormeaux
Côte-Saint-Luc
YUL
Pointe-Claire
Montréal-Ouest
5 km
3 mi
Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen

Places that are a **#Fromagerie Hamel** this is work in progress

See **all** the #brands

```
{'query': {'filtered': {'filter': {'and': [{'bool': {'must_not': [{'exists': {'field': 'edtf:depr0.02996301651 seconds
ecated'}}]}}]},
                    'query': {'term': {'wof:brand_id': 1126128733}}}}}
```

Would you like try this query again to include records that have been deprecated ?

the map is currently centered at 45.544799, -73.575879 #11

1126128741 Fromagerie Hamel — this is a venue in **Canada** that we believe **to be open**

it is tagged cheese and atwatermarket / it was last modified on **July 12**

152276073 Fromagerie Hamel — this is a venue in **Canada** that we believe **to be**

filter by **#brand**

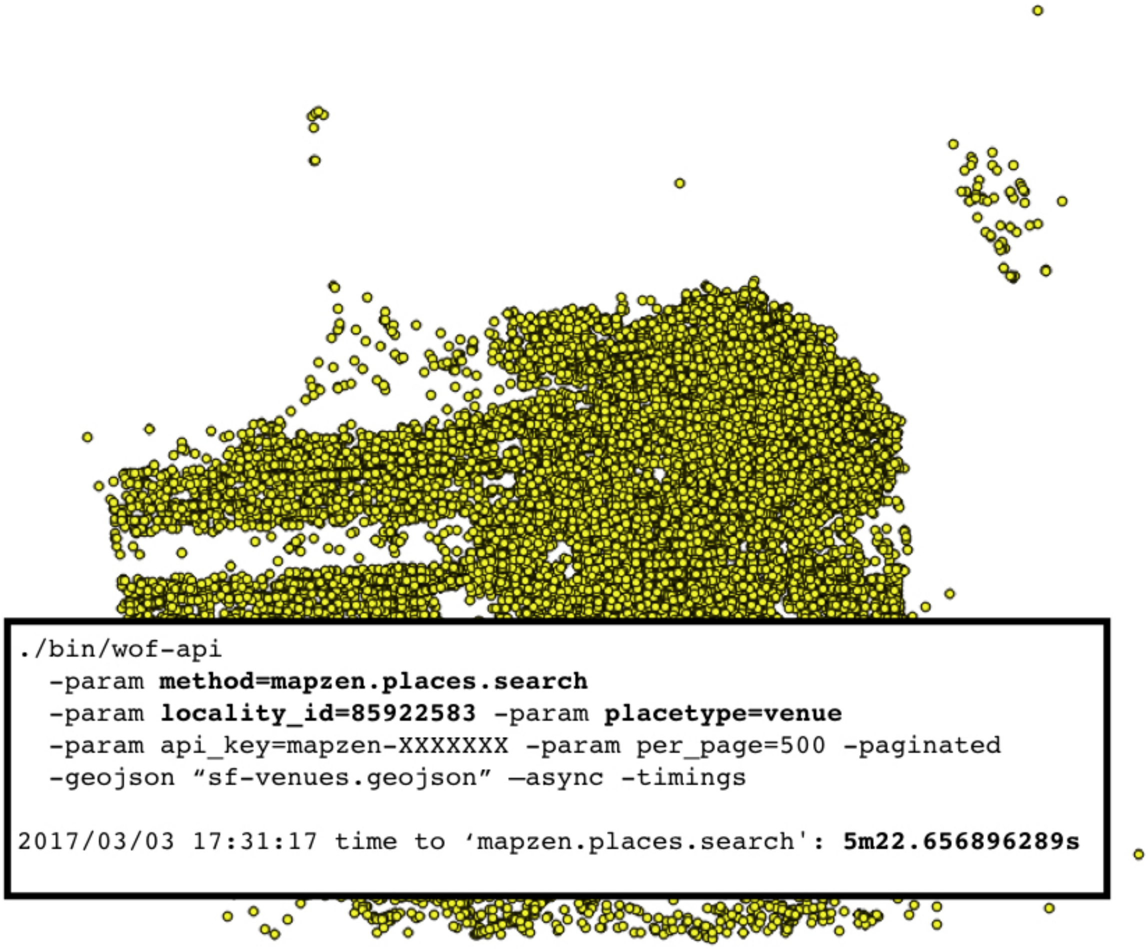
Fromagerie Hamel 4

filter by **category**

(https://mapzen.com/blog/whosonfirst-sotmus-2017#brands)

It's early days still but **we are also starting to do the work to track brands (https://github.com/whosonfirst-data/whosonfirst-brands/)**, assigning them each their own stable IDs and static records. It's very primitive work right now and relies entirely on unsophisticated string-matching but it's something to build on.

One of the medium-term goals we have is to work with actual brands and to have them contribute and take responsibility for maintaining their venues inside of Who's On First.

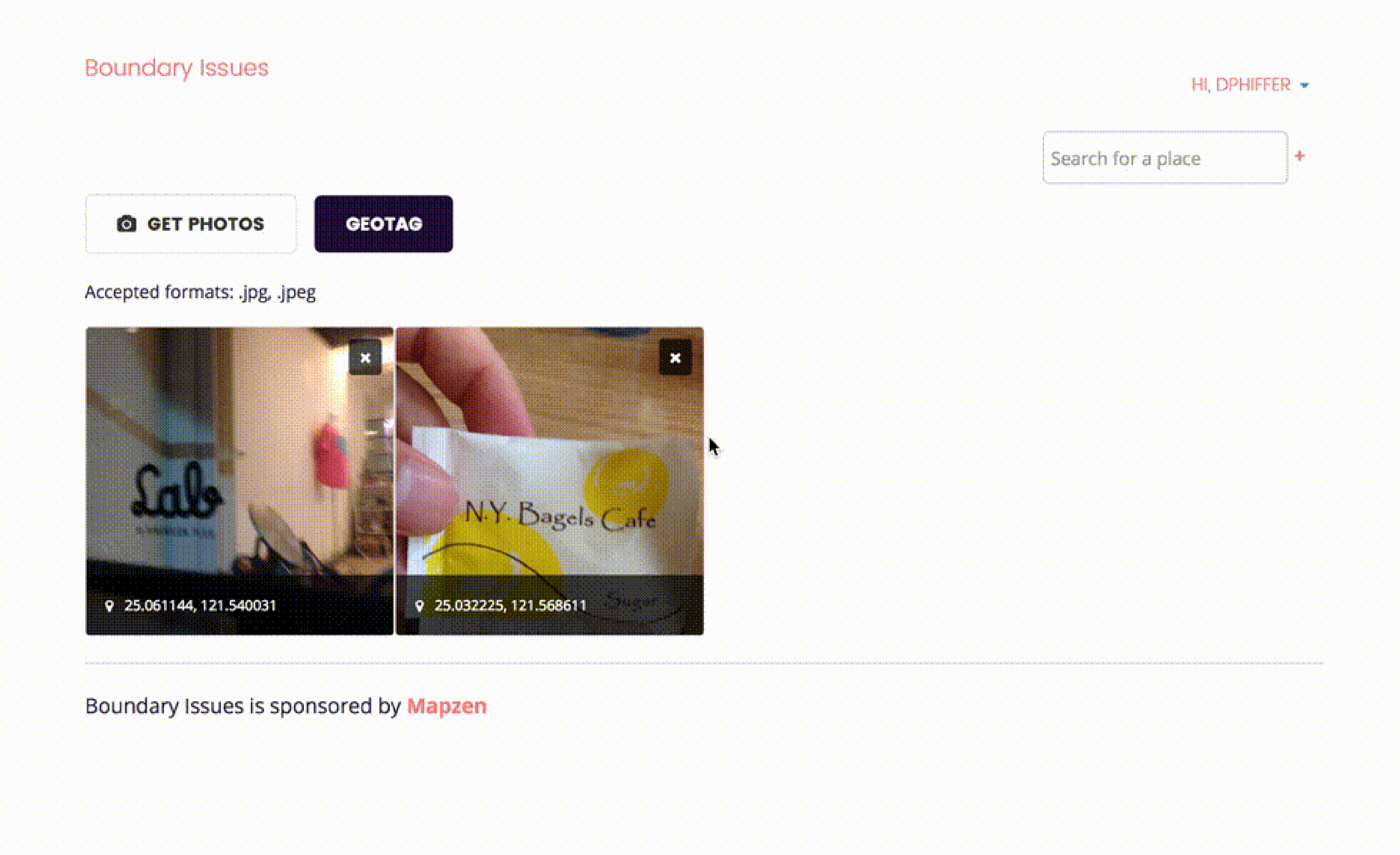


```
./bin/wof-api
  -param method=mapzen.places.search
  -param locality_id=85922583 -param placetype=venue
  -param api_key=mapzen-XXXXXXX -param per_page=500 -paginated
  -geojson “sf-venues.geojson” —async -timings

2017/03/03 17:31:17 time to ‘mapzen.places.search': 5m22.656896289s
```

(https://mapzen.com/blog/whosonfirst-sotmus-2017#api)

In addition to static downloads everything is available through the **Mapzen Places API (https://mapzen.com/documentation/places/)**.

It takes about 5 minutes to **grab all the venues in San Francisco (https://github.com/whosonfirst/go-whosonfirst-api)** and dump them out as a GeoJSON FeatureCollection.


Boundary Issues
HI, DPHIFFER
Search for a place
GET PHOTOS
GEOTAG
Accepted formats: .jpg, .jpeg
25.061144, 121.540031
25.032225, 121.568611
Boundary Issues is sponsored by Mapzen

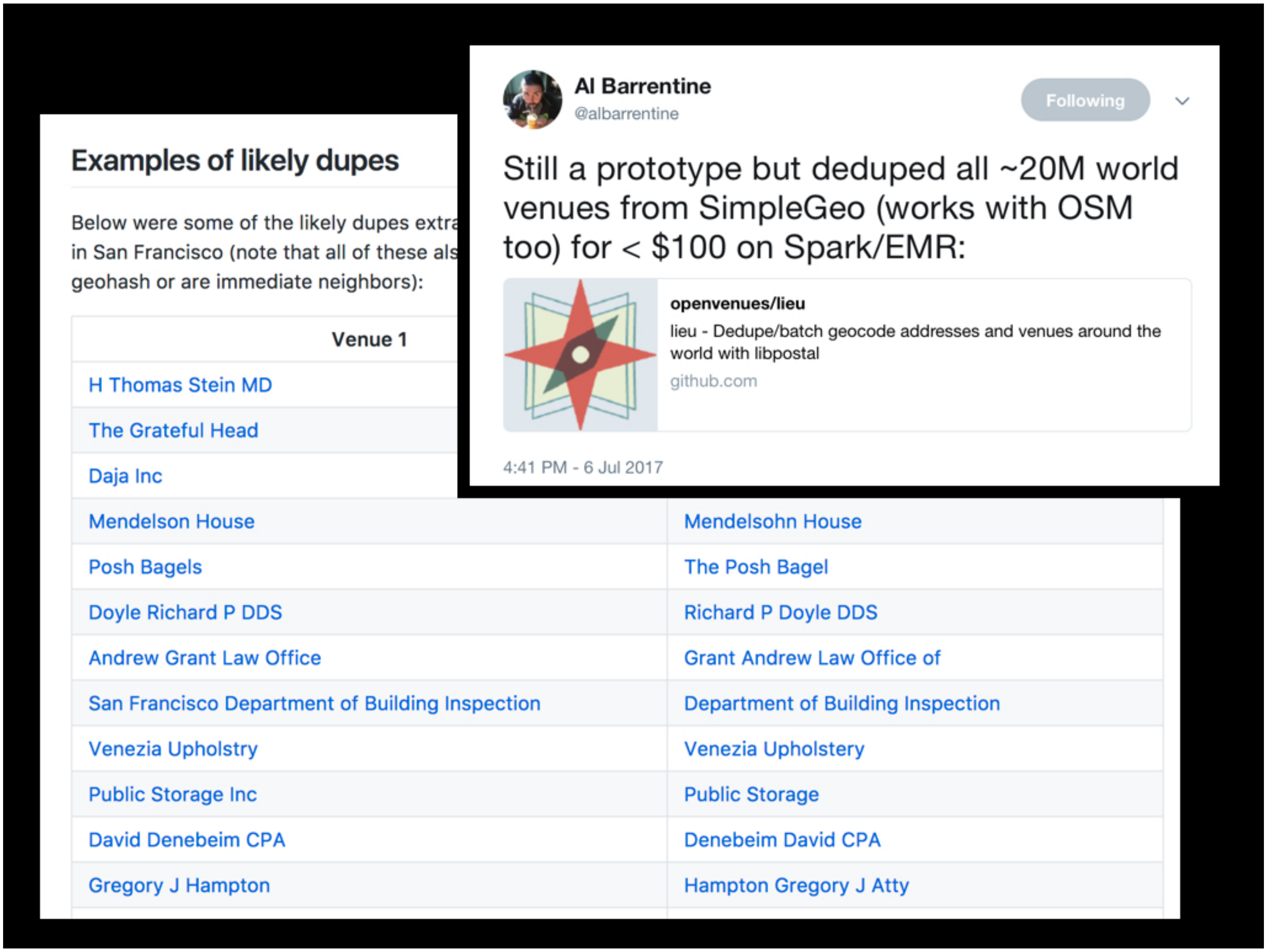
(https://mapzen.com/blog/whosonfirst-sotmus-2017#geotagging)

I could have easily spent the entire talk discussing **the UI and UX challenges (https://mapzen.com/blog/simple-is-hard)** around editing the existing data and making it fast and easy to add new data. We are a small team so our biggest challenge is simply the short number of hours in the day.

This is work by another colleague, **Dan Phiffer (https://phiffer.org/)**, to think about how we might **use geotagged photos to add new venues (https://mapzen.com/blog/geotagging-wof-venues/)**.

Dan spent the summer in **South Korea (https://places.mapzen.com/id/85632231)** and **Taiwan (https://places.mapzen.com/id/85632403)**, places where we have (had) no venue data and where all the Mapzen services like tiles and web-based APIs grind to a halt and become unusable when you're traveling on a throttled data plan.

Dan's tool allowed him to roam freely during the day quickly taking pictures of venues and points of interest and uploading them after the fact to create new records in Who's On First.



Examples of likely dupes

Below were some of the likely dupes extra
in San Francisco (note that all of these als
geohash or are immediate neighbors):

Venue 1	
H Thomas Stein MD	
The Grateful Head	
Daja Inc	
Mendelson House	Mendelsohn House
Posh Bagels	The Posh Bagel
Doyle Richard P DDS	Richard P Doyle DDS
Andrew Grant Law Office	Grant Andrew Law Office of
San Francisco Department of Building Inspection	Department of Building Inspection
Venezia Upholstry	Venezia Upholstery
Public Storage Inc	Public Storage
David Denebeim CPA	Denebeim David CPA
Gregory J Hampton	Hampton Gregory J Atty

Al Barrentine
@albarrentine

Still a prototype but deduped all ~20M world venues from SimpleGeo (works with OSM too) for < $100 on Spark/EMR:

openvenues/lieu
lieu - Dedupe/batch geocode addresses and venues around the world with libpostal
github.com

4:41 PM - 6 Jul 2017

(https://mapzen.com/blog/whosonfirst-sotmus-2017#lieu)

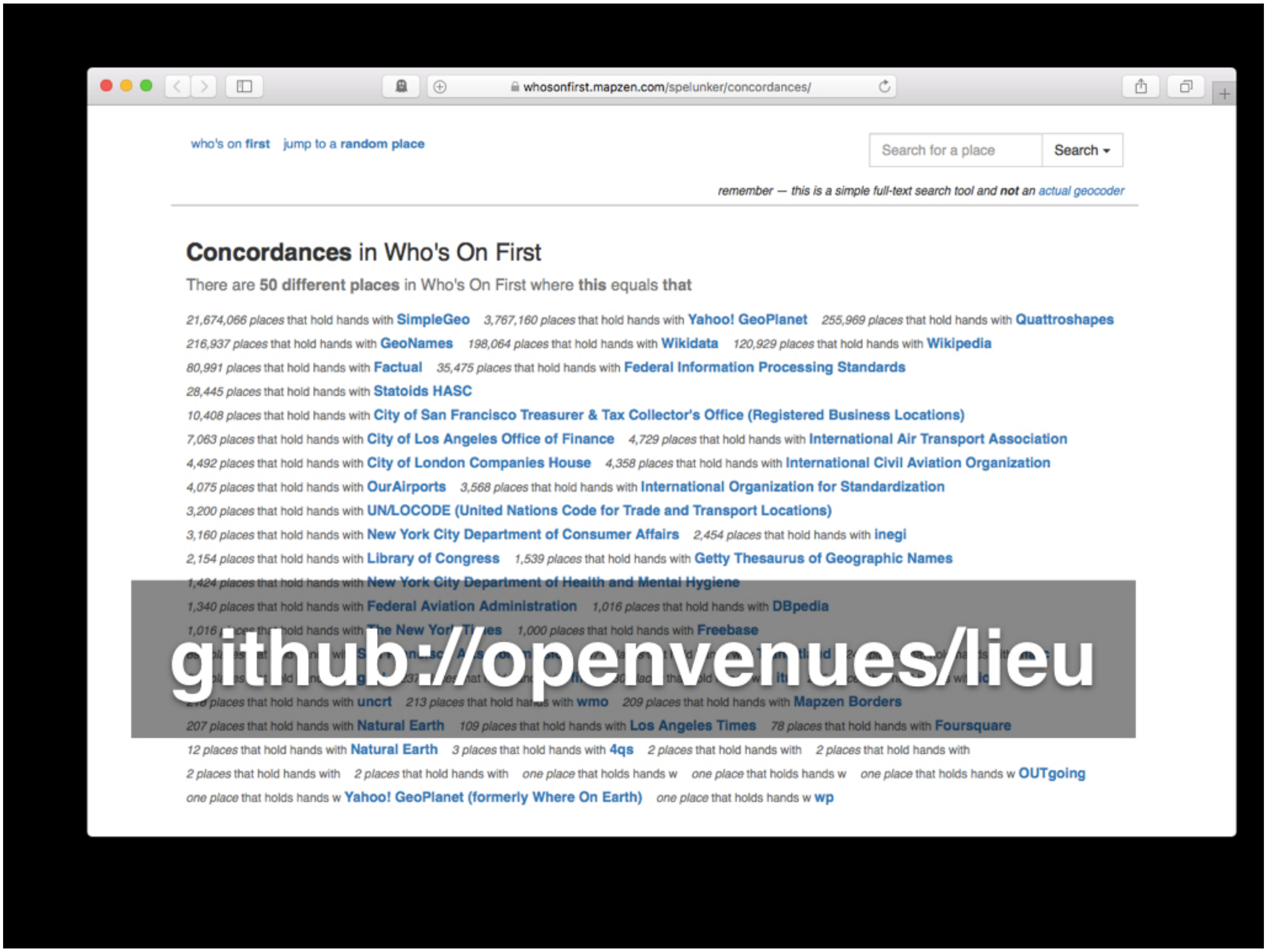
Finally we’ve been working with **Al Barrentine (https://twitter.com/albarrentine)**, who wrote the **libpostal address parser (https://mapzen.com/blog/inside-libpostal/)**. Building on that work we’ve asked Al to take a first pass at address de-duplication.

If address parsing is where you go to cry then address de-duplication is where you go to give up.

As we go forward and start to import newer datasets it is clear that the first challenge will be reconciling what we already have in Who’s On First with whatever new offering is on the table.

We don’t have the choice of giving up on this problem.

Al’s remit was not to solve all the things but to build a viable “1.0” tool that demonstrated tangible results and can be improved upon going forward.



remember — this is a simple full-text search tool and **not** an actual geocoder

Concordances in Who's On First

There are **50 different places** in Who's On First where **this** equals **that**

21,674,066 places that hold hands with **SimpleGeo** *3,767,160 places* that hold hands with **Yahoo! GeoPlanet** *255,969 places* that hold hands with **Quattroshapes**
216,937 places that hold hands with **GeoNames** *198,064 places* that hold hands with **Wikidata** *120,929 places* that hold hands with **Wikipedia**
80,991 places that hold hands with **Factual** *35,475 places* that hold hands with **Federal Information Processing Standards**
28,445 places that hold hands with **Statoids HASC**
10,408 places that hold hands with **City of San Francisco Treasurer & Tax Collector's Office (Registered Business Locations)**
7,063 places that hold hands with **City of Los Angeles Office of Finance** *4,729 places* that hold hands with **International Air Transport Association**
4,492 places that hold hands with **City of London Companies House** *4,358 places* that hold hands with **International Civil Aviation Organization**
4,075 places that hold hands with **OurAirports** *3,568 places* that hold hands with **International Organization for Standardization**
3,200 places that hold hands with **UN/LOCODE (United Nations Code for Trade and Transport Locations)**
3,160 places that hold hands with **New York City Department of Consumer Affairs** *2,454 places* that hold hands with **inegi**
2,154 places that hold hands with **Library of Congress** *1,539 places* that hold hands with **Getty Thesaurus of Geographic Names**
1,424 places that hold hands with **New York City Department of Health and Mental Hygiene**
1,340 places that hold hands with **Federal Aviation Administration** *1,016 places* that hold hands with **DBpedia**
1,016 places that hold hands with **The New York Times** *1,000 places* that hold hands with **Freebase**
[illegible]
[illegible]
[illegible] places that hold hands with **uncrt** *213 places* that hold hands with **wmo** *209 places* that hold hands with **Mapzen Borders**
207 places that hold hands with **Natural Earth** *109 places* that hold hands with **Los Angeles Times** *78 places* that hold hands with **Foursquare**
12 places that hold hands with **Natural Earth** *3 places* that hold hands with **4qs** *2 places* that hold hands with *2 places* that hold hands with
2 places that hold hands with *2 places* that hold hands with *one place* that holds hands w *one place* that holds hands w *one place* that holds hands w **OUTgoing**
one place that holds hands w **Yahoo! GeoPlanet (formerly Where On Earth)** *one place* that holds hands w **wp**

github://openvenues/lieu

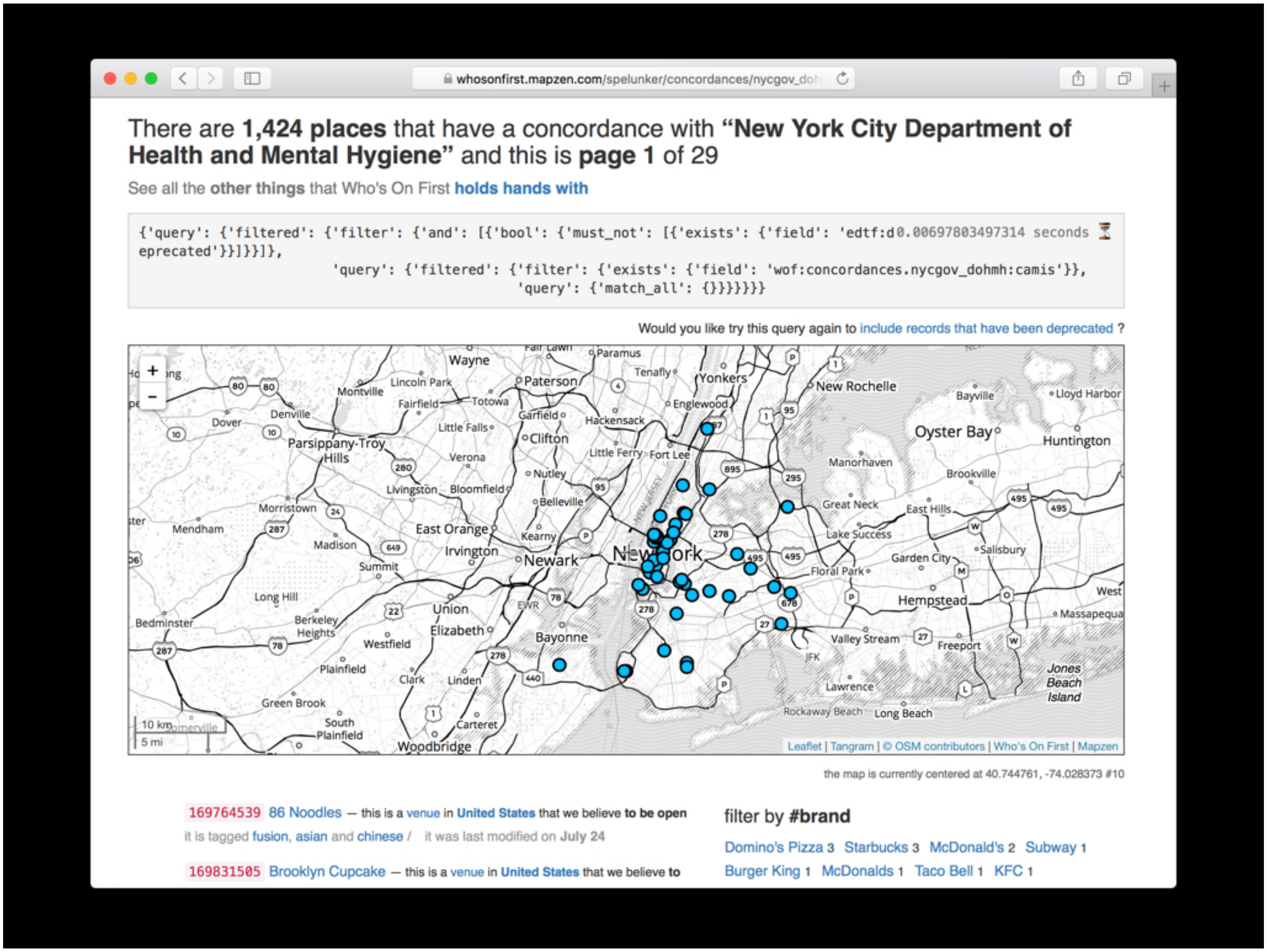
(https://mapzen.com/blog/whosonfirst-sotmus-2017#business)

The tool is called `lieu` and, as with libpostal **it is available as open source software (https://github.com/openvenues/lieu)**.

As Al has been working on things I've been testing address de-duplication against Who's On First and publicly available business listings from the usual suspects: **New York (https://whosonfirst.mapzen.com/spelunker/concordances/nycgov_dca/)**, **San Francisco (https://whosonfirst.mapzen.com/spelunker/concordances/sfgov_rbl/)**, **Los Angeles (https://whosonfirst.mapzen.com/spelunker/concordances/lacity_oof/)** and **London (https://whosonfirst.mapzen.com/spelunker/concordances/companieshouse/)**.

One of the benefits of Al's software is that when we establish a match between of our venues and an official listing we can flag that venue as **"current" (https://whosonfirst.mapzen.com/spelunker/placetypes/venue/?is_current=1)** and remove some of the ambiguity that might surround it. We can add **concordances from Who's On First (https://whosonfirst.mapzen.com/spelunker/concordances)** back to the city and their venue records.

The next step as we get ready to finish up this phase of work and push it out the nest is to de-duplicate Who's On First against itself.



There are **1,424 places** that have a concordance with **"New York City Department of Health and Mental Hygiene"** and this is **page 1** of 29

See all the **other things** that Who's On First **holds hands with**

```
{'query': {'filtered': {'filter': {'and': [{'bool': {'must_not': [{'exists': {'field': 'edtf:d0.00697803497314 seconds
eprecated'}}]}}]},
                     'query': {'filtered': {'filter': {'exists': {'field': 'wof:concordances.nycgov_dohmh:camis'}},
                                  'query': {'match_all': {}}}}}}}
```

Would you like try this query again to include records that have been deprecated ?

Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen
10 km
5 mi

the map is currently centered at 40.744761, -74.028373 #10

169764539 86 Noodles — this is a venue in **United States** that we believe **to be open**
it is tagged fusion, asian and chinese / it was last modified on **July 24**

169831505 Brooklyn Cupcake — this is a venue in **United States** that we believe **to**

filter by **#brand**

Domino's Pizza 3 Starbucks 3 McDonald's 2 Subway 1
Burger King 1 McDonalds 1 Taco Bell 1 KFC 1

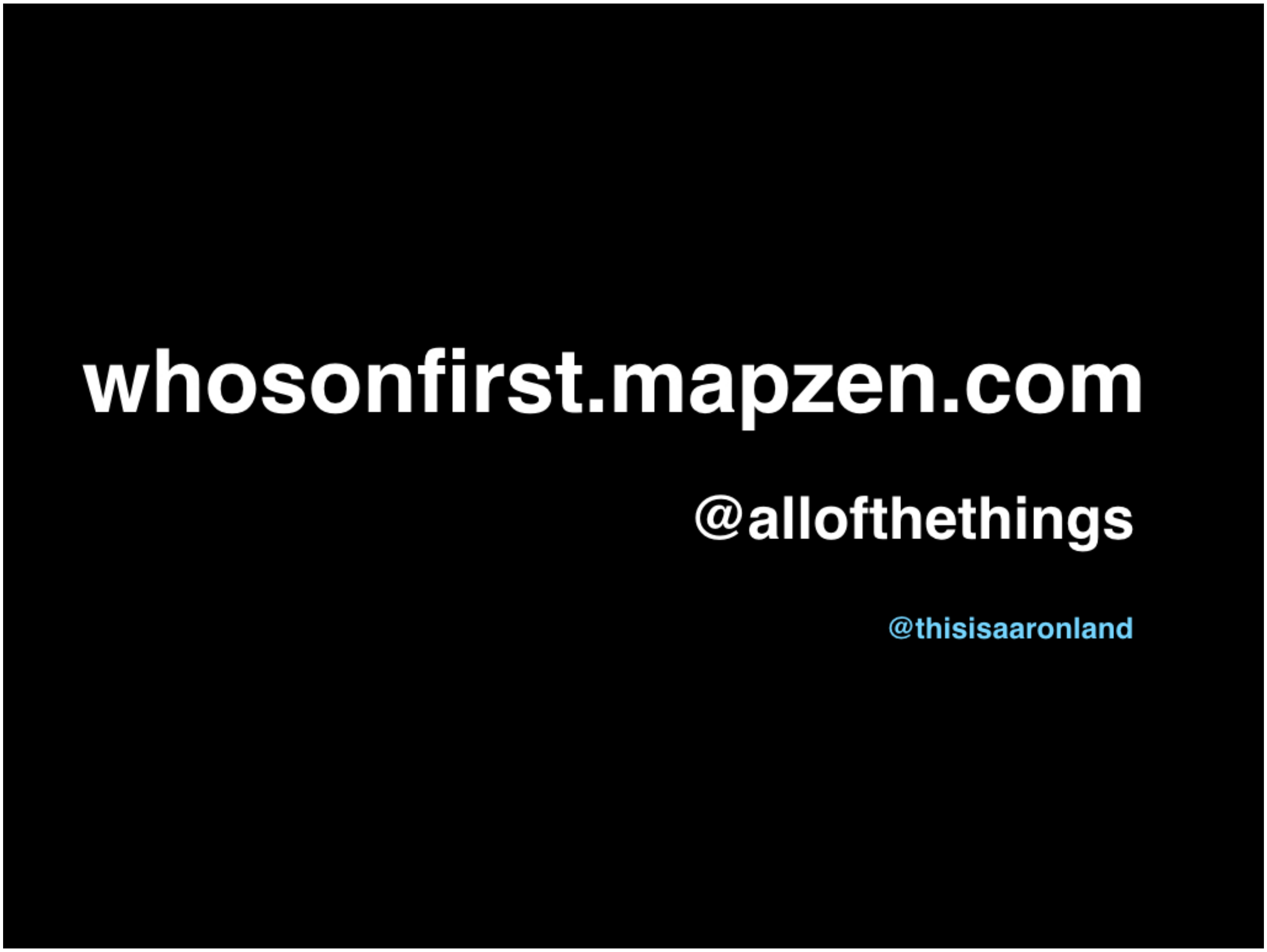
(https://mapzen.com/blog/whosonfirst-sotmus-2017#fuzzy)

The software is also able to work with fuzzy and approximate location data for venues.

For example, it's possible to use the latitude and longitude for the postal code associated with an address, in the absence of coordinate data for the address itself, and lieu will still find successful matches.

This means that basically **every health and safety inspection record ever published in the US (https://whosonfirst.mapzen.com/spelunker/concordances/nycgov_dohmh/)** is fair-game for de-duping.

We can validate records that we already have and import new ones with a high, if not perfect, degree of confidence and start chipping away at the "no new stuff" problem.


whosonfirst.mapzen.com

@allofthethings

@thisisaaronland

(https://mapzen.com/blog/whosonfirst-sotmus-2017#outro)

By now some of you may be wondering what this has to do with **OpenStreetMap (https://www.openstreetmap.org/)**? The answer is: I don’t know but something, I hope.

If Who’s On First can’t use OpenStreetMap data then we would like to make sure that it’s easy for the OpenStreetMap community, if you choose, to use our data. Maybe that means all we do is establish and maintain concordances between the two projects and **maybe it means more (https://stevecoast.com/2015/09/30/license-ascent/)**.

My hope, at the end of this talk, is that I will have raised enough interest and enough questions that we might find out together.

Thank you.

Aaron’s talk is ***now on YouTube (https://www.youtube.com/watch?v=GUizIw0LNjc?rel=0)****!*

Photo credits:

- ***https://commons.wikimedia.org/wiki/File:C2E2_2015_-_Wonder_Twins_(17118576890).jpg (https://commons.wikimedia.org/wiki/File:C2E2_2015_-_Wonder_Twins_(17118576890).jpg)***

· 24 October 2017 ·

Aaron Straup Cope

Aaron is happiest in the presence of olive oil.

© 2017 Mapzen

State of the Map US 2017 - OSMLR and Who's on First and You

osm (/tag/osm) whosonfirst (/tag/whosonfirst) places (/tag/places)

Thanks to everyone who stopped by our table at State of the Map US in Boulder! It's a great reminder of both the power of open data and the community behind it.

Videos of this year's talks **are now online (https://www.youtube.com/playlist?list=PLqjPa29lMiE2k2Sp5L5rb6ntduG9dt0te)**!

Some talks we found particularly interesting:

- **Mapping Sidewalks for Pedestrian Routing (TriMet (Portland), Madeline Steele) (https://youtu.be/w64pLD_bgKk)**
- **Bicycle Network Analysis - OSM for Cycling (People for Bikes, Spencer Gardner) (https://youtu.be/YgyynQDPQnQ)**
- **OSM in the Age of Autonomous Vehicles (https://youtu.be/13jSjGV6Tyg)**

- **Emily Jacobi's Keynote (https://www.youtube.com/watch?v=WJlT8li_tRw&t=21s)**

You can choose to **watch our very own Aaron discuss Who's On First 🤝OpenStreetMap (https://www.youtube.com/watch?v=GUizIw0LNjc)**, or **read his words and slides at your leisure (https://mapzen.com/blog/whosonfirst-sotmus-2017/)**.

Dave Nesbitt gave a nice talk on OSMLR (Stable Linear-Referencing System for OSM) (now an **Amazon Public Datasets (https://mapzen.com/blog/osmlr-released-as-public-dataset/)**!)

You can learn much more about OSMLR from our **blog posts (https://mapzen.com/tag/osmlr/)**, along with **Dave's slides (https://s3.amazonaws.com/mapzen-assets/images/sotmus-2017/OSMLR_SoTM17v1.pdf)**.

See you next year!

Preview image by Kristen DiLuca

· 27 October 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

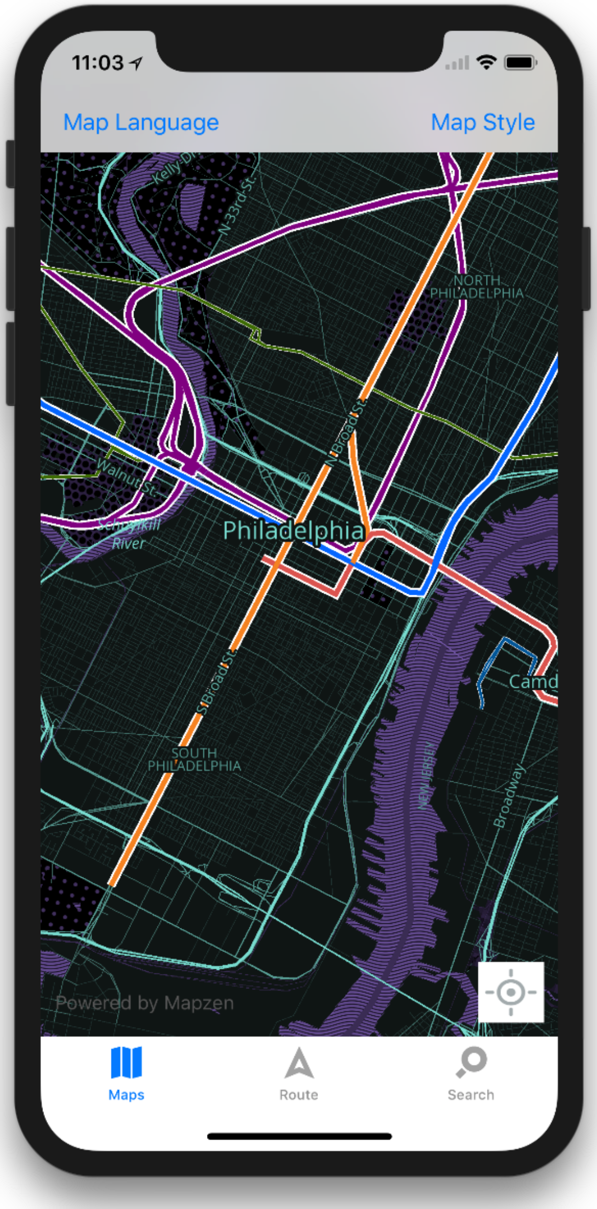
Themes come to Mobile!

mobile (/tag/mobile) android (/tag/android) tangram (/tag/tangram) cartography (/tag/cartography)

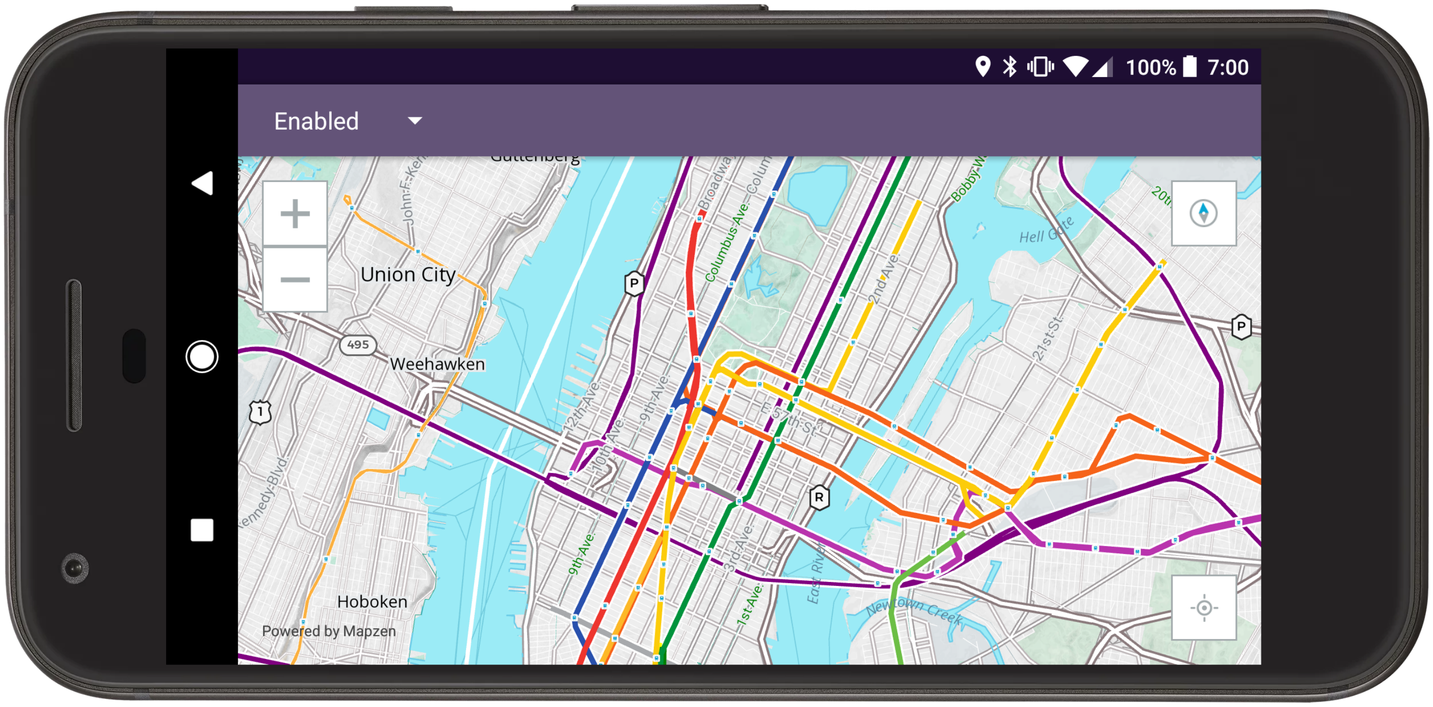
We’ve been hard at work on the Mobile team preparing a bunch of new features. We’d like to highlight a few of them for you today!

Themes

As you might already know, **Themes (https://mapzen.com/blog/refill-themes-part-2/)** are a way to easily customize your Tangram maps without diving into the YAML style sheets. Our Mobile SDKs on Android and iOS now support color, label, and detail level themes across all our styles.


11:03
Map Language
Map Style
Kelly Dr
N 33rd St
NORTH PHILADELPHIA
N Broad St
Walnut St
Schuylkill River
Philadelphia
Camd
S Broad St
SOUTH PHILADELPHIA
Broadway
Powered by Mapzen
Maps
Route
Search

We'll be continuing to upgrade the functionality in these areas as we add additional theme options to our **house styles (https://mapzen.com/products/maps/)**, so keep an eye here and on our **iOS (https://github.com/mapzen/ios/releases)** and **Android (https://github.com/mapzen/android/releases)** Github releases for more information!


100% 7:00
Enabled
Union City
Weehawken
Hoboken
Powered by Mapzen
Hell Gate
Newtown Creek
Columbus-Ave
Broadway
2nd-Ave
1st-Ave
3rd-Ave
9th-Ave
10th-Ave
12th-Ave
21st-St
East River
495

Custom Style Sheets (Experimental)

One of the biggest requests we've heard from our customers is to be able to support custom style sheets in our Mobile SDKs. Our work on themes has laid the groundwork for this customization in your iOS and Android applications. Note that this support is experimental and will likely change as we evolve the API, but we're releasing it early to gather feedback from the community. Take a look at it on **iOS (https://github.com/mapzen/ios/blob/master/MapzenSDK/Themes.swift)** and **Android (https://github.com/mapzen/android/blob/master/core/src/main/java/com/mapzen/android/graphics/model/ThemedMapStyle.java)** and let us know your thoughts by opening a Github issue!

512 Pixel Tile Support

As part of most Mobile SDK releases, we've upgraded the Mapzen style sheets and Tangram-es to their latest respective versions, thus enabling 512 pixel vector tile support. Take a look at the **blog post (https://mapzen.com/blog/512-pixel-tiles-and-basemaps/)** about how to enable this functionality.

Note that this functionality is *opt-in*, and that you can do so at run time by using our scene update functionality. In the future we'll be making this an option available on our custom style sheet functionality.

If you're using the built-in Mapzen style sheets, this functionality is already enabled for you - no work needs to be done!

Get started

There are many more feature enhancements, upgrades, and bug fixes in the works and on the way. Feel free to check out the latest releases for our **Android (https://github.com/mapzen/android)** and **iOS (https://github.com/mapzen/ios)** repositories. As always, if you have any questions, reach out at **hello@mapzen.com (mailto:hello@mapzen.com)** or open a github issue in the appropriate repo!

· 01 November 2017 ·

Matt Smollinger

Matt would 3D print an SDK if it were possible.

Sarah Lensing

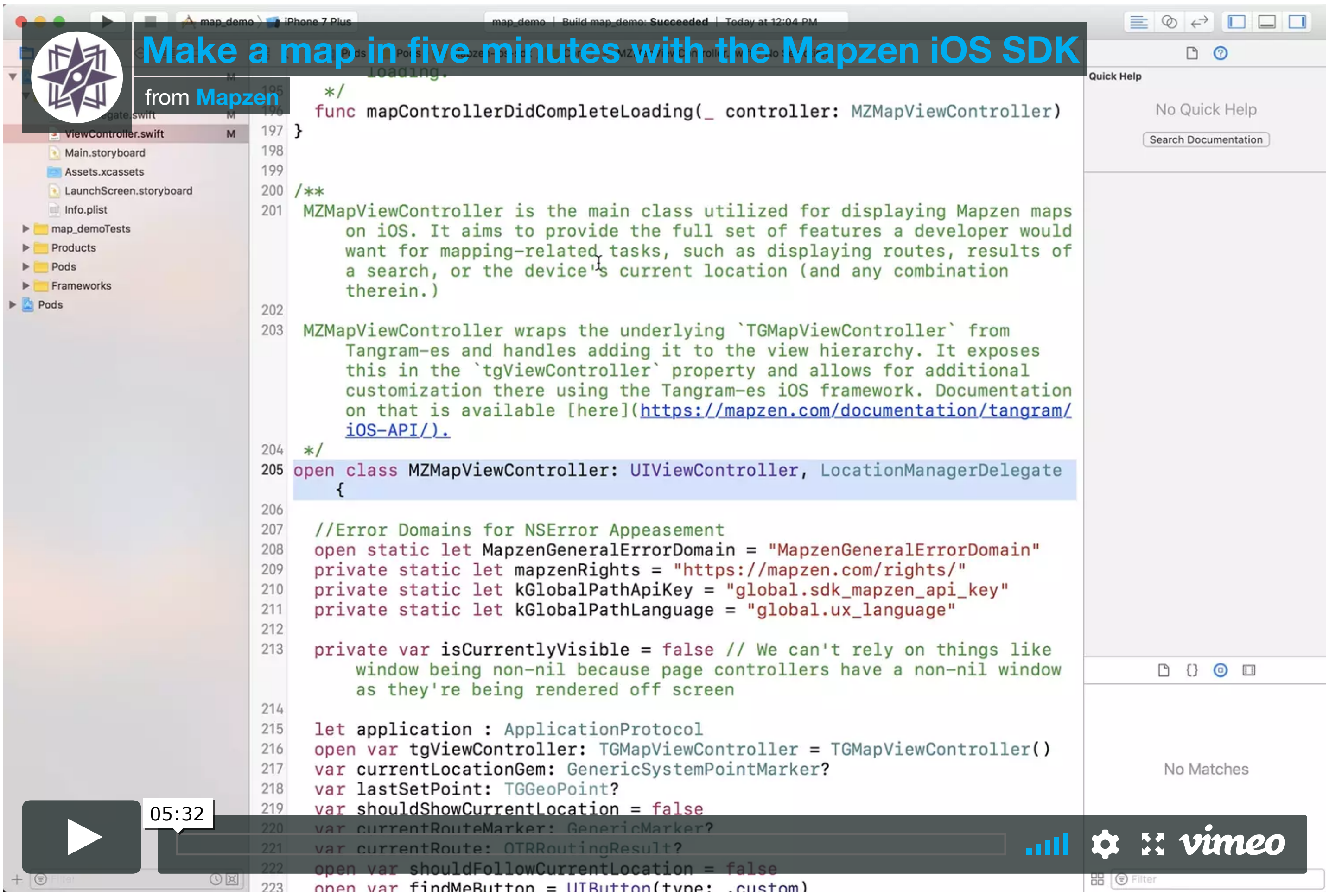
mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

© 2017 Mapzen

Make a map in 5 minutes with the Mapzen iOS SDK!

mobile (/tag/mobile) tangram (/tag/tangram) tutorial (/tag/tutorial)

We made a video for our iOS SDK to help developers get up and running faster – get a map in an app in five minutes!


Make a map in five minutes with the Mapzen iOS SDK
from Mapzen
map_demo
iPhone 7 Plus
map_demo | Build map_demo: Succeeded | Today at 12:04 PM
ViewController.swift M
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
map_demoTests
Products
Pods
Frameworks
Pods
Quick Help
No Quick Help
Search Documentation
No Matches
05:32
vimeo

```
        loading.
     */
    func mapControllerDidCompleteLoading(_ controller: MZMapViewController)
}


/**
 MZMapViewController is the main class utilized for displaying Mapzen maps
     on iOS. It aims to provide the full set of features a developer would
     want for mapping-related tasks, such as displaying routes, results of
     a search, or the device's current location (and any combination
     therein.)

 MZMapViewController wraps the underlying `TGMapViewController` from
     Tangram-es and handles adding it to the view hierarchy. It exposes
     this in the `tgViewController` property and allows for additional
     customization there using the Tangram-es iOS framework. Documentation
     on that is available [here](https://mapzen.com/documentation/tangram/
     iOS-API/).
 */
open class MZMapViewController: UIViewController, LocationManagerDelegate
    {

  //Error Domains for NSError Appeasement
  open static let MapzenGeneralErrorDomain = "MapzenGeneralErrorDomain"
  private static let mapzenRights = "https://mapzen.com/rights/"
  private static let kGlobalPathApiKey = "global.sdk_mapzen_api_key"
  private static let kGlobalPathLanguage = "global.ux_language"

  private var isCurrentlyVisible = false // We can't rely on things like
      window being non-nil because page controllers have a non-nil window
      as they're being rendered off screen

  let application : ApplicationProtocol
  open var tgViewController: TGMapViewController = TGMapViewController()
  var currentLocationGem: GenericSystemPointMarker?
  var lastSetPoint: TGGeoPoint?
  var shouldShowCurrentLocation = false
  var currentRouteMarker: GenericMarker?
  var currentRoute: OTRRoutingResult?
  open var shouldFollowCurrentLocation = false
  open var findMeButton = UIButton(type: .custom)
```

Make a map in five minutes with the Mapzen iOS SDK from Mapzenon Vimeo(**https://vimeo.com/235418187 (https://vimeo.com/235418187)**)

You can also read along with the tutorial at **https://mapzen.com/documentation/ios/getting-started/ (https://mapzen.com/documentation/ios/getting-started/)**

You can also get started in five minute **on Android too (https://mapzen.com/blog/get-started-with-android-in-five-minutes)**! Let us know what other videos you might like to see!

· 03 November 2017 ·

Matt Smollinger

Matt would 3D print an SDK if it were possible.

Sarah Lensing

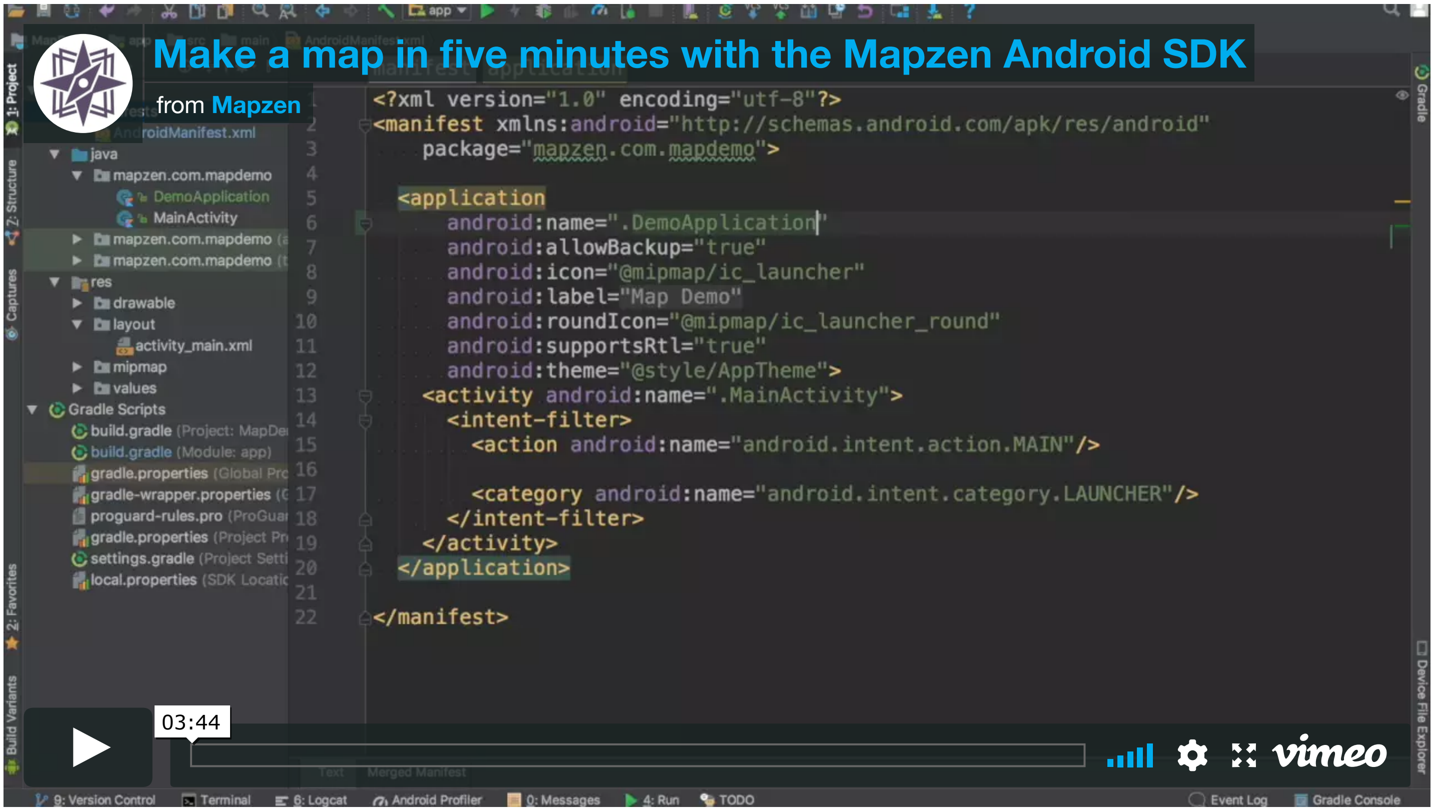
mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

© 2017 Mapzen

Make a map in 5 minutes with the Mapzen Android SDK!

mobile (/tag/mobile) tangram (/tag/tangram) tutorial (/tag/tutorial)

You can use our **Android SDK (https://mapzen.com/documentation/android/getting-started/)** to get up and running faster – get a map in an app in five minutes!


Make a map in five minutes with the Mapzen Android SDK
from Mapzen
java
mapzen.com.mapdemo
DemoApplication
MainActivity
res
drawable
layout
activity_main.xml
mipmap
values
Gradle Scripts
03:44
vimeo

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mapzen.com.mapdemo">

    <application
        android:name=".DemoApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Map Demo"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Make a map in five minutes with the Mapzen Android SDKfrom Mapzen on Vimeo (https://vimeo.com/241205494)

You can also read along with the tutorial at **https://mapzen.com/documentation/android/getting-started/ (https://mapzen.com/documentation/android/getting-started/)**

Don't forget our **iOS SDK (https://mapzen.com/blog/get-started-with-ios-in-five-minutes/)** five minute video! Any other videos you might like to see? Drop us a line at **hello@mapzen.com (mailto:hello@mapzen.com)**!

· 07 November 2017 ·

Matt Smollinger

Matt would 3D print an SDK if it were possible.

Sarah Lensing

mobile developer. startup junkie. athlete. older sister. nyu alum. vermonter. maker of beef jerky and other random food items.

© 2017 Mapzen

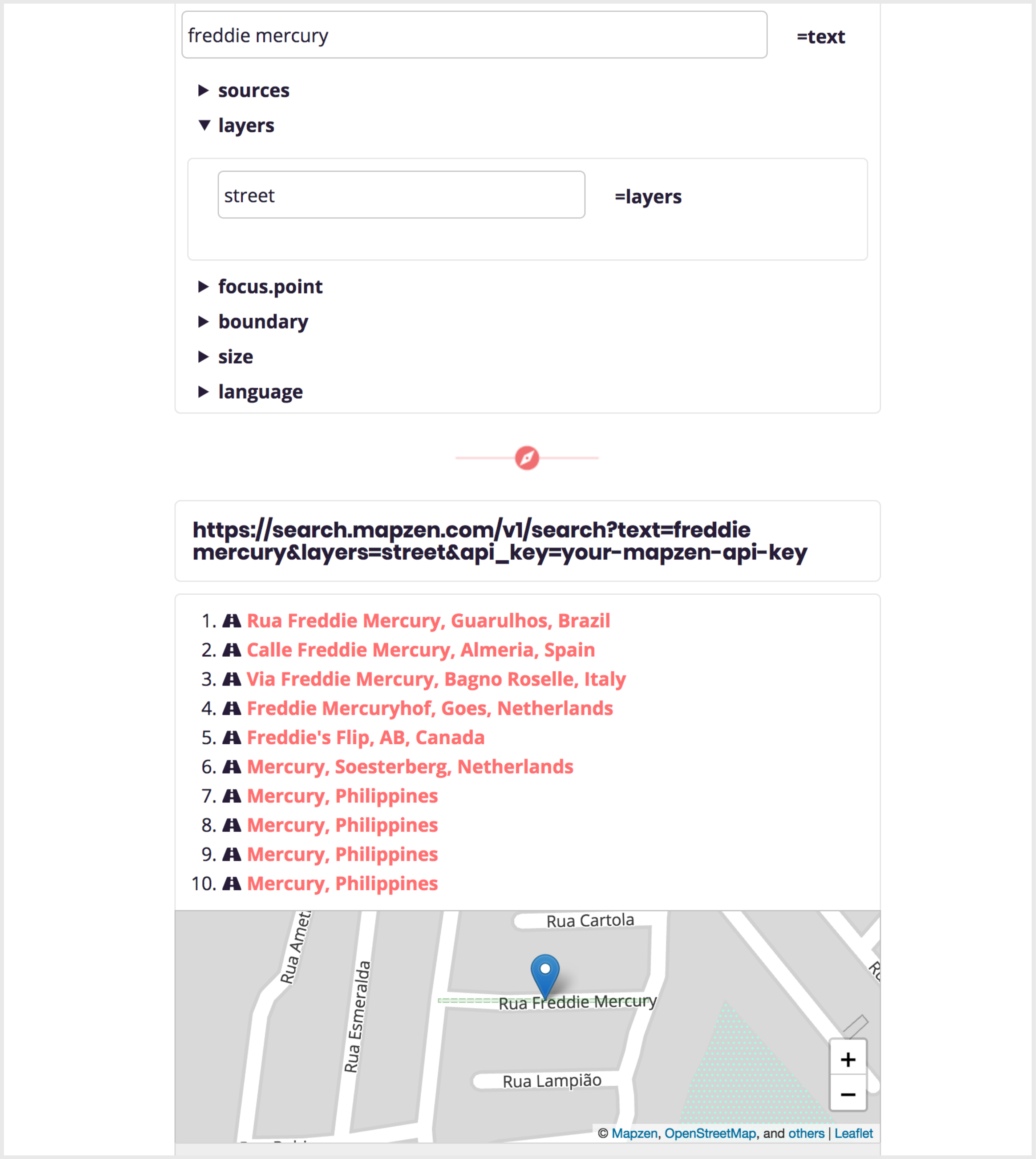
Mapzen Search Explorer

search (/tag/search) tutorial (/tag/tutorial)

Mapzen Search has **lots of API endpoints (https://mapzen.com/documentation/search/search-workflows/)**: `/search`, `/search/structured`, `/reverse`, `/autocomplete`, `/place`. We've previously talked about **which endpoint to use when (https://mapzen.com/blog/which-search-and-when/)**, but what about all those parameters?

In addition to our **search documentation (https://mapzen.com/documentation/search)**, the **Mapzen Search Explorer (https://mapzen.com/search/explorer)** has the answers you seek. This interactive tool demonstrates how parameters like `sources`, `layers`, `focus.point`, `boundary`, `language` (and others) can help you filter and focus search results for each endpoint.

As you type, we generate a URL for our API based on the parameters you've selected. Once you've built the query, we show you a list of results and points on a map. (You don't need an API key to explore! However, **you will need to add one (https://mapzen.com/developers)** if you use the API request URL in your app or map.)



freddie mercury =text

▶ sources
▼ layers

street =layers

▶ focus.point
▶ boundary
▶ size
▶ language

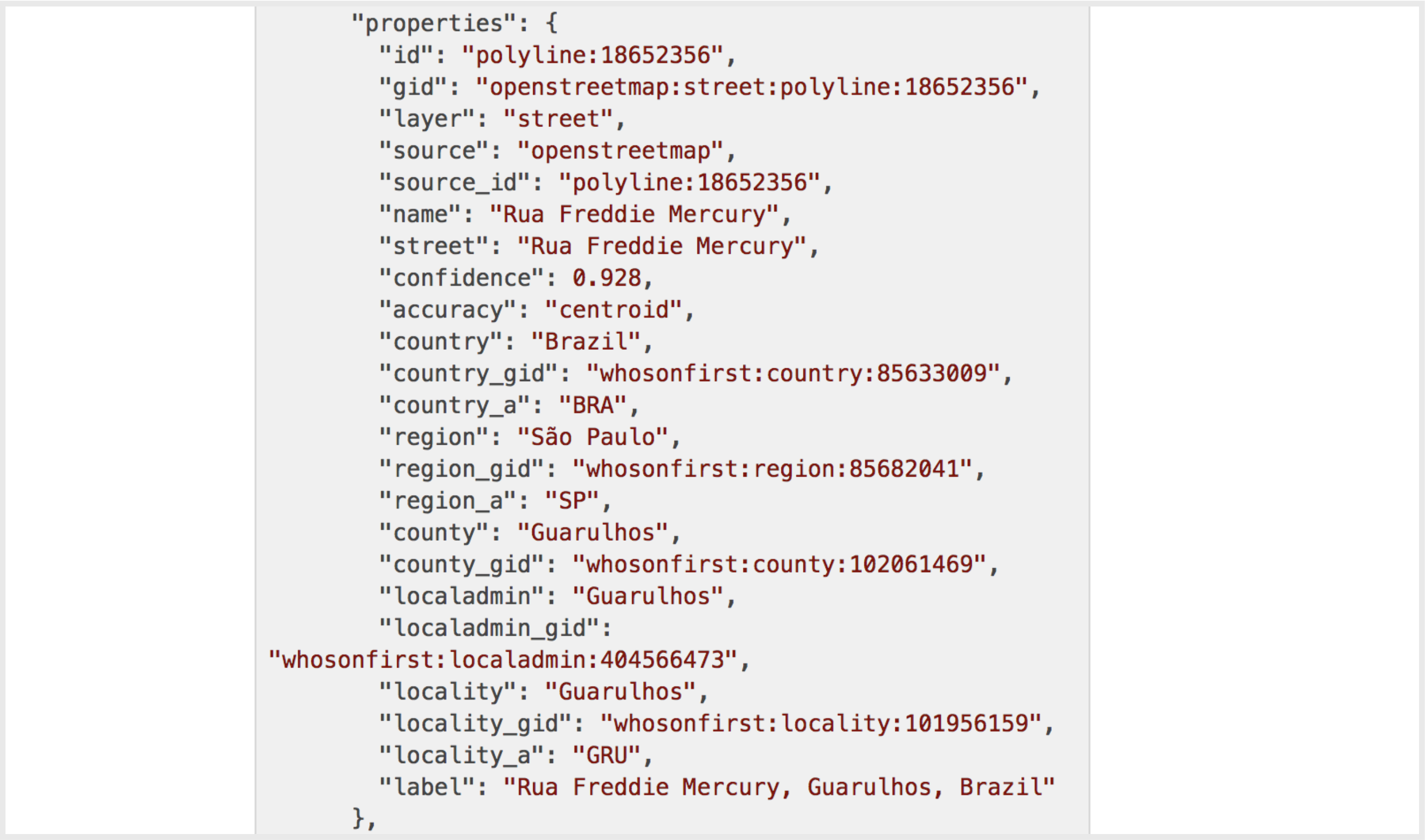
https://search.mapzen.com/v1/search?text=freddie mercury&layers=street&api_key=your-mapzen-api-key

1. Rua Freddie Mercury, Guarulhos, Brazil
2. Calle Freddie Mercury, Almeria, Spain
3. Via Freddie Mercury, Bagno Roselle, Italy
4. Freddie Mercuryhof, Goes, Netherlands
5. Freddie's Flip, AB, Canada
6. Mercury, Soesterberg, Netherlands
7. Mercury, Philippines
8. Mercury, Philippines
9. Mercury, Philippines
10. Mercury, Philippines

Rua Amet
Rua Esmeralda
Rua Cartola
Rua Freddie Mercury
Rua Lampião
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

https://mapzen.com/search/explorer/?query=search&text=freddie%20mercury&layers=street (https://mapzen.com/search/explorer/?query=search&text=freddie%20mercury&layers=street)

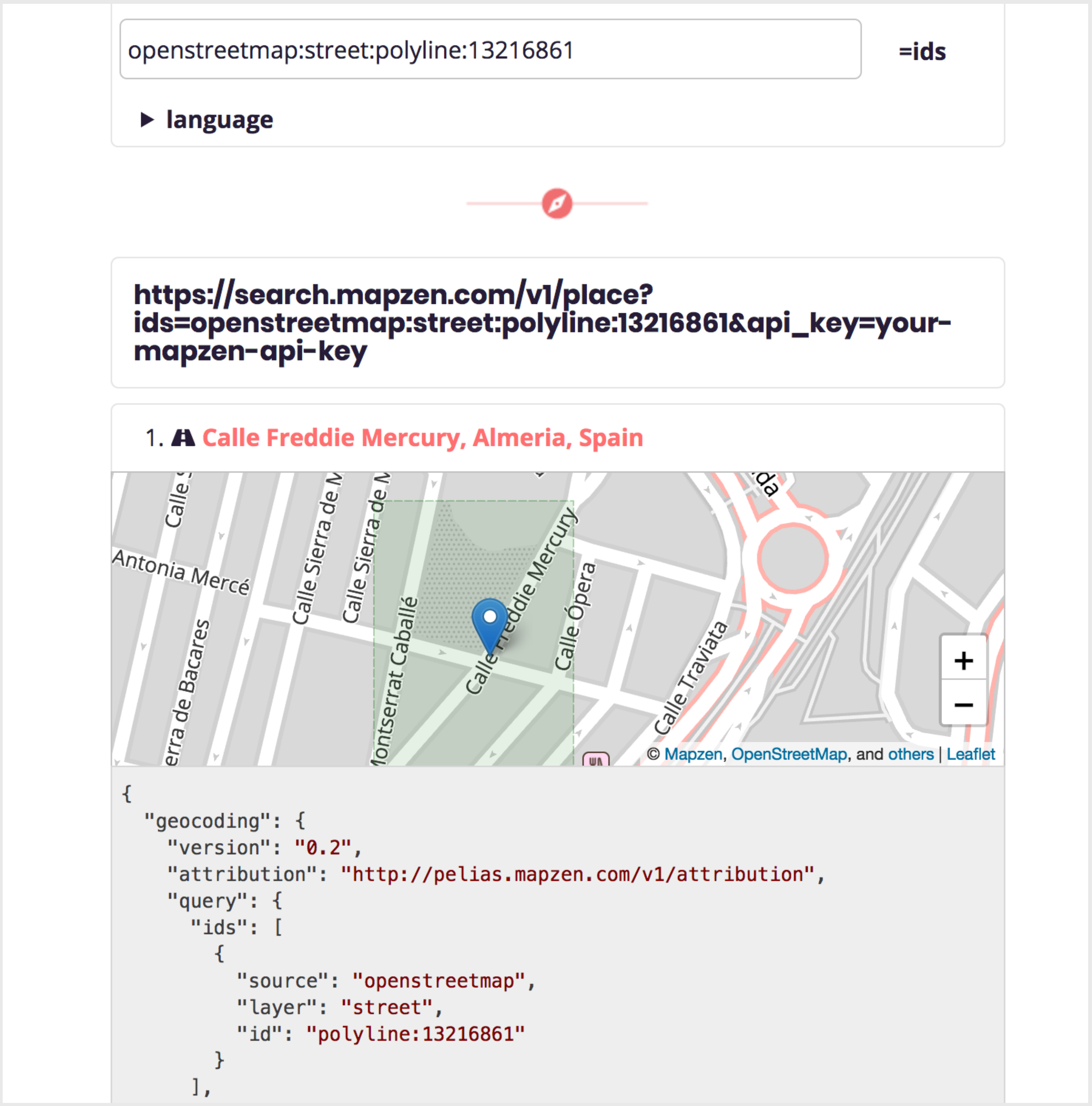
We also show the API response of all the results as GeoJSON. This includes search metadata and confidence scores.



```
      "properties": {
        "id": "polyline:18652356",
        "gid": "openstreetmap:street:polyline:18652356",
        "layer": "street",
        "source": "openstreetmap",
        "source_id": "polyline:18652356",
        "name": "Rua Freddie Mercury",
        "street": "Rua Freddie Mercury",
        "confidence": 0.928,
        "accuracy": "centroid",
        "country": "Brazil",
        "country_gid": "whosonfirst:country:85633009",
        "country_a": "BRA",
        "region": "São Paulo",
        "region_gid": "whosonfirst:region:85682041",
        "region_a": "SP",
        "county": "Guarulhos",
        "county_gid": "whosonfirst:county:102061469",
        "localadmin": "Guarulhos",
        "localadmin_gid":
"whosonfirst:localadmin:404566473",
        "locality": "Guarulhos",
        "locality_gid": "whosonfirst:locality:101956159",
        "locality_a": "GRU",
        "label": "Rua Freddie Mercury, Guarulhos, Brazil"
      },
```

https://mapzen.com/search/explorer/?query=search&text=freddie%20mercury&size=40&layers=street (https://mapzen.com/search/explorer/?query=search&text=freddie%20mercury&size=40&layers=street)

You can click on an individual result and get more information about its source via our `place` endpoint.


openstreetmap:street:polyline:13216861
=ids
▶ language

https://search.mapzen.com/v1/place?ids=openstreetmap:street:polyline:13216861&api_key=your-mapzen-api-key

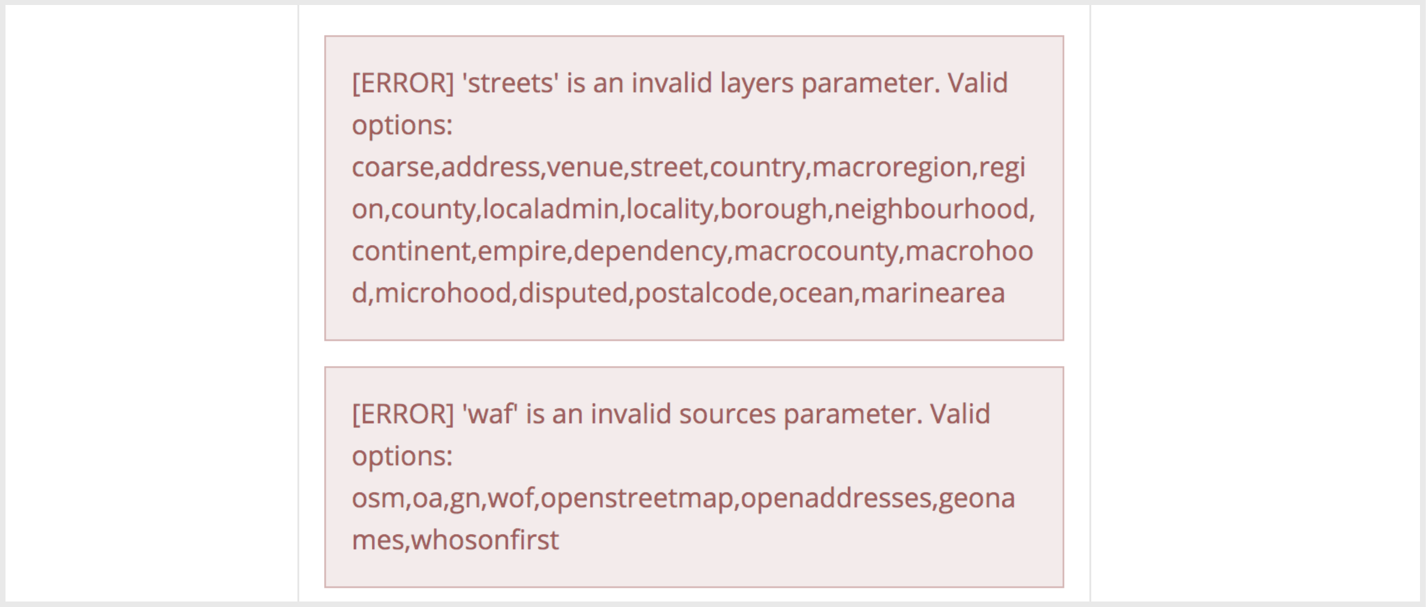
1. Calle Freddie Mercury, Almeria, Spain

Antonia Mercé
Calle Sierra de M
Calle Sierra de M
erra de Bacares
Montserrat Caballé
Calle Freddie Mercury
Calle Ópera
Calle Traviata
+
−
© Mapzen, OpenStreetMap, and others | Leaflet

```
{
  "geocoding": {
    "version": "0.2",
    "attribution": "http://pelias.mapzen.com/v1/attribution",
    "query": {
      "ids": [
        {
          "source": "openstreetmap",
          "layer": "street",
          "id": "polyline:13216861"
        }
      ],
```

https://mapzen.com/search/explorer/?query=place&ids=openstreetmap%3Astreet%3Apolyline%3A18652356 (https://mapzen.com/search/explorer/?query=place&ids=openstreetmap%3Astreet%3Apolyline%3A18652356)

Meaningful error messages are shown inline, helping you learn more about the parameters and what they do (and don’t do).



[ERROR] 'streets' is an invalid layers parameter. Valid options: coarse,address,venue,street,country,macroregion,region,county,localadmin,locality,borough,neighbourhood,continent,empire,dependency,macrocounty,macrohood,microhood,disputed,postalcode,ocean,marinearea

[ERROR] 'waf' is an invalid sources parameter. Valid options: osm,oa,gn,wof,openstreetmap,openaddresses,geonames,whosonfirst

https://mapzen.com/search/explorer/?query=search&text=%E2%80%8ERick%20Astley&sources=waf&layers=streets (https://mapzen.com/search/explorer/?query=search&text=%E2%80%8ERick%20Astley&sources=waf&layers=streets)

The Search Explore is a convenient way to share a query and API request with coworkers and friends. It’s a handy troubleshooting tool if you can’t find something you expect (or do find something you don’t expect). If you’re having an issue and can reproduce it with the Search Explorer, you get a unique URL that you can share with us.

Take the **Search Explorer (https://mapzen.com/search/explorer)** for a spin and let us know what you find!

· 16 November 2017 ·

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

© 2017 Mapzen

NACIS 2017 videos are online!

vector-tiles (/tag/vector-tiles) data (/tag/data) cartography (/tag/cartography) routing (/tag/routing) mobile (/tag/mobile) data (/tag/data)

If you weren't able to make it to NACIS in Montreal this year, **the first round of videos of the talks are now online (https://www.youtube.com/channel/UCxleHCzc6YynGbzJMWx6C7w)**!

We found a few talks especially interesting:

- Victoria Johnson, **A Field Guide to 20th Century Airline Route Maps (https://www.youtube.com/watch?v=9cpaxfv5EXA)**
- Lauren Tierney, **National Geographic Magazine's Cartography of Antarctica (https://www.youtube.com/watch?v=c_QRdWVZDTI)**
- Anton Thomas, **Drawing a Continent By Hand (https://www.youtube.com/watch?v=B20WKWnDmRQ)**

Mapzen had a number of speakers across a breadth of topics:

Tuesday, Oct 10 - Practical Cartography Day (https://nacis2017.sched.com/event/BDzV/practical-cartography-day-early-morning-sessions-journee-cartographie-pratique-seances-du-matin)

Ekta Daryanani - **The UX of Navigation: A Road Trip from NYC to NACIS (https://www.youtube.com/watch?v=7tY5UMycNfY)**

Let's take a drive from NYC to Montreal and think about what your typical navigation app shows and tells you to do as you drive. The objective is to never have to glance at a screen for more than a couple of seconds, coz accidents!

Hanbyul Jo - **Getting Crafty with Vector Tiles (https://www.youtube.com/watch?v=J7L5UvIAtf8)**

I have been making tools for digital fabrications (3d prints, pen plots, laser cuts) with map vector tiles. I want to show what kind of fabrication is possible and the tools that people can use for their own crafts.

Nathaniel Vaughn Kelso - **Back to the Desktop (https://www.youtube.com/watch?v=OZ9LgXrt-ug)**

Vector maps have enabled a new generation of map rendering and dynamic styling for the web and mobile applications. But how can you get at those vectors and use them in desktop map projects? I'll demonstrate a direct to Illustrator workflow and another using QGIS and Mapzen's vector tile data.

Wednesday, October 11 - Cartographic Applications II (https://nacis2017.sched.com/event/BDzV/practical-cartography-day-early-morning-sessions-journee-cartographie-pratique-seances-du-matin)

Aaron Staup Cope - **Who's On First Two Years Later - Theory, Practice and Gotchas (https://www.youtube.com/watch?v=WbRRrYEgtRU)**

The Who's On First (WOF) project was launched by Mapzen in 2015 with the goal of being a robust and openly-licensed gazetteer with global coverage spanning continents to (20 million and counting) venues, and everything in between. WOF is used by Mapzen for many of its own services but has been designed and architected to be portable, durable and as infrastructure-agnostic as possible.

This presentation will outline the project's original motivations and its current state of affairs, discuss some of the challenges we've faced along the way and review what we're focusing on next.

(We turned Aaron's talk **into a blog post (https://mapzen.com/blog/whosonfirst-nacis-2017/)**.)

Geraldine Sarmiento's lightning talk on **Visual Language in Cartography (https://nacis2017.sched.com/event/B95I/lightning-talks-i-presentations-eclairs-i)** is not up yet, but in the meantime, **explore lines and forms with the Morphology tool (https://mapzen.com/blog/morphology/)**!

See you next year!

· 21 November 2017 ·

Mapzen

Opening maps with open source and open data.

© 2017 Mapzen

Polyline Precision

routing (/tag/routing)

When you send a request to the Mapzen Mobility APIs, part of the response you receive may appear to be nothing more than a jumble of characters. However, this is actually something known as an encoded polyline that stores a series of latitude, longitude coordinates as a single string, and greatly reduces the size of the shape.

This polyline…

```
 unoagAn dnhFsVxCkAsQUeEkAaRyBq\\_Ewm@mE_r@UwDeAuOMiCkAqQcB{UcF{u@OgCsA_Tly@}J~b@cFzP
yBvC_@~C]jA]t@O C?
xBNz@N~DrFje@ro@xBvClUb[tx@lhAjVb\lh@ls@`fAbyAj[fc@|EpGW\}DbGyLtOm@l@u@\cANkA?
aDL_DNgD\iVhCqRhCeEuo@OkA?]G]OyBoCwc@e@qG
```

…encodes these points:

```
 [[-122.407448,37.789947],[-122.407525,37.790325],[-122.407227,37.790363],
[-122.407128,37.790374],[-122.406823,37.790412],[-122.40635,37.790473],
[-122.405602,37.790569],[-122.404786,37.790672],[-122.404694,37.790683],
[-122.404427,37.790718],[-122.404358,37.790725],[-122.404061,37.790763],
[-122.403695,37.790813],[-122.402817,37.790927],[-122.402749,37.790935],
[-122.402413,37.790977],[-122.402222,37.790042],[-122.402108,37.789466],
[-122.402047,37.78918],[-122.402031,37.789104],[-122.402016,37.789024],
[-122.402001,37.788986],[-122.401993,37.788959],[-122.401993,37.788894],
[-122.402001,37.788833],[-122.402009,37.788803],[-122.402131,37.788707],
[-122.402909,37.788093],[-122.402985,37.788032],[-122.403435,37.787673],
[-122.40461,37.78675],[-122.405076,37.786376],[-122.405915,37.785713],
[-122.407357,37.784576],[-122.407937,37.784122],[-122.408074,37.784011],
[-122.408089,37.784023],[-122.408219,37.784118],[-122.408486,37.784339],
[-122.408509,37.784362],[-122.408524,37.784389],[-122.408532,37.784423],
[-122.408532,37.784461],[-122.408539,37.784542],[-122.408547,37.784622],
[-122.408562,37.784706],[-122.408631,37.785079],[-122.4087,37.785392],
```

~~~
[-122.407921,37.785491],[-122.407883,37.785499],[-122.407868,37.785499],
[-122.407853,37.785503],[-122.407792,37.785511],[-122.407204,37.785583],
[-122.407067,37.785602]]```
~~~

What you see when the precision is off

One of our most common technical support requests is from users attempting to encode or decode a route response with fewer than six digits of precision. Doing this can result in nonsensical coordinates, negative elevations values, or points showing up in the ocean—far away from potential route paths.

For example, a user working with Map Matching recently noticed, *“I always get error 171: No suitable edges near location, and some weird positions like [378.07744, -1224.19701], [378.07744, -1224.19756]…”* It turned out the user had decoded with a polyline utility from Google, and the problems were fixed by switching to Mapzen’s code.

Why six digits of decimal degree precision?

A key distinction of how Mapzen APIs integrate encoded polylines is that they use **six** digits of **decimal degree precision (https://en.wikipedia.org/wiki/Decimal_degrees)**. This is an enhancement of the **algorithms from Google (https://developers.google.com/maps/documentation/utilities/polylinealgorithm)** and the APIs from other mapping providers that use ***five*** digits of precision. That additional digit gives you extra detail and can return more accurate results along a path — at the equator, six digits gives you 10cm of precision, while five digits yields about a meter. This can make a significant difference in the placement of a route on a road. Also, GeoJSON recently standardized on 6 digits of precision.

The requirement for using six digits applies to any API that accepts inputs of or returns encoded polylines, including Mapzen Turn-by-Turn, Isochrone, Optimized Route, Map Matching, and Elevation. These use calculations from **Valhalla (https://github.com/valhalla)**, the open-source routing project.

Why does a difference in precision cause issues? The last step in decoding a polyline involves converting an integer into a decimal coordinate, with the presumption that it will be a certain number of digits long. Using five digits of decimal precision instead of six in this last step will lead to the decimal point being placed in the wrong location.

If you have trouble remembering the number of digits with Mapzen APIs, these rhymes can help:

Fewer than six, you need a fix.

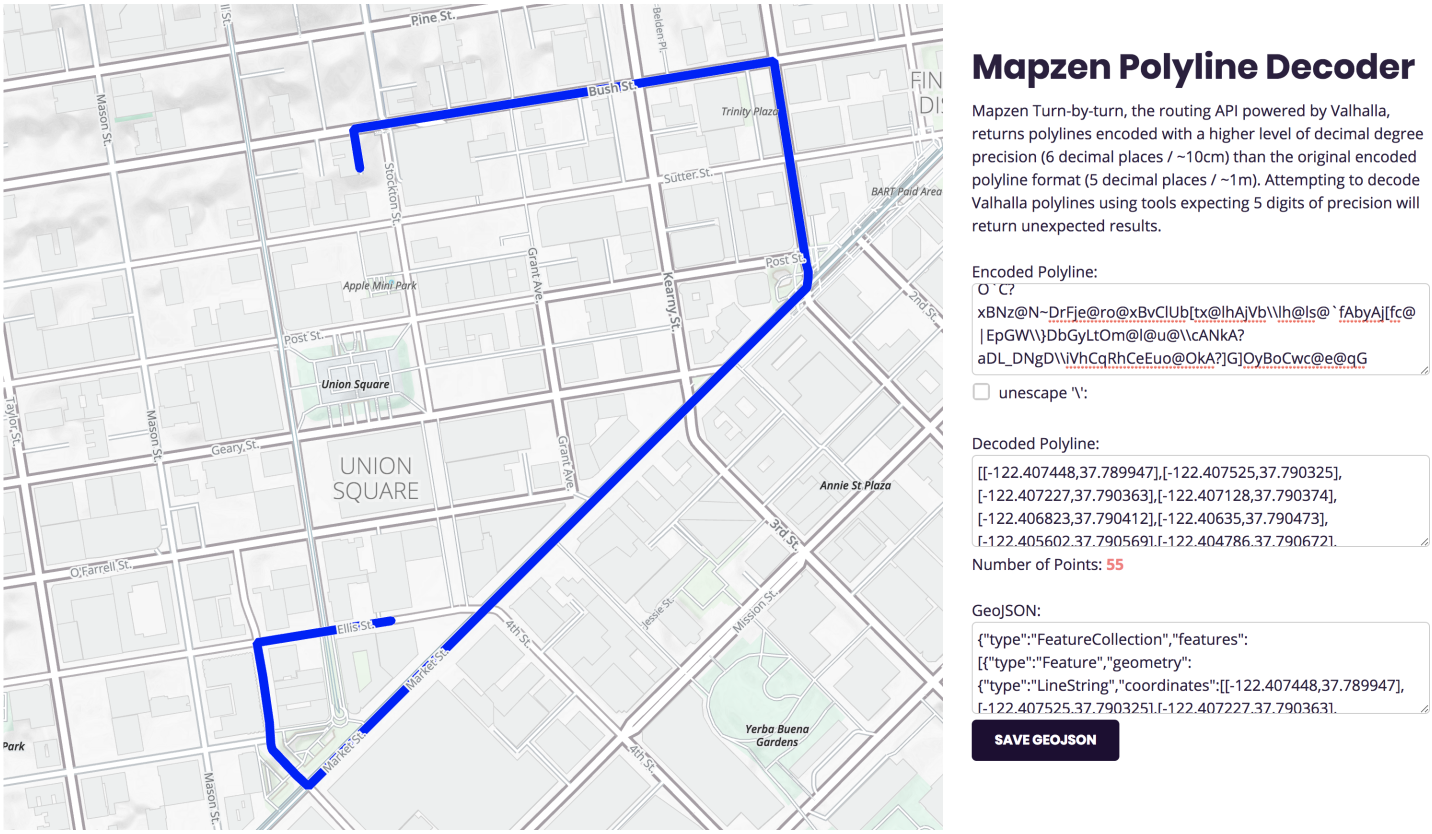
With only five, results take a dive.

Use five plus one, you are done.

Decode with our tools for test, results are best.

Tools for decoding with six digits

When you are working with Mapzen's APIs, only use our tools to decode polylines. In the **Mobility documentation (https://mapzen.com/documentation/mobility/decoding/)**, you can find sample code to decode shapes in JavaScript, C++, and Python. We also built **this handy tool (http://mapzen.github.io/marketing/demos/polyline/index.html)** where you can paste an encoded polyline string, decode it, and see the locations on a map (and save to GeoJSON).


Pine St.
Bush St.
Trinity Plaza
Sutter St.
BART Paid Area
Post St.
Stockton St.
Grant Ave.
Kearny St.
Mason St.
Apple Mini Park
Union Square
UNION SQUARE
Taylor St.
Geary St.
O'Farrell St.
Ellis St.
Market St.
4th St.
3rd St.
2nd St.
Jessie St.
Mission St.
Annie St Plaza
Yerba Buena Gardens
Park

Mapzen Polyline Decoder

Mapzen Turn-by-turn, the routing API powered by Valhalla, returns polylines encoded with a higher level of decimal degree precision (6 decimal places / ~10cm) than the original encoded polyline format (5 decimal places / ~1m). Attempting to decode Valhalla polylines using tools expecting 5 digits of precision will return unexpected results.

Encoded Polyline:

```
O`C?
xBNz@N~DrFje@ro@xBvClUb[tx@lhAjVb\\lh@ls@`fAbyAj[fc@
|EpGW\\}DbGyLtOm@l@u@\\cANkA?
aDL_DNgD\\iVhCqRhCeEuo@OkA?]G]OyBoCwc@e@qG
```

☐ unescape '\':

Decoded Polyline:

```
[[-122.407448,37.789947],[-122.407525,37.790325],
[-122.407227,37.790363],[-122.407128,37.790374],
[-122.406823,37.790412],[-122.40635,37.790473],
[-122.405602,37.790569],[-122.404786,37.790672],
```

Number of Points: 55

GeoJSON:

```
{"type":"FeatureCollection","features":
[{"type":"Feature","geometry":
{"type":"LineString","coordinates":[[-122.407448,37.789947],
[-122.407525,37.790325],[-122.407227,37.790363],
```

SAVE GEOJSON

· 01 December 2017 ·

Rhonda Glennon

Rhonda is Mapzen's technical publications manager and writes about maps and developer tools.

John Oram

Product marketing, burrito quality assurance, 4D map tiler. One day John will make as many maps as he writes about.

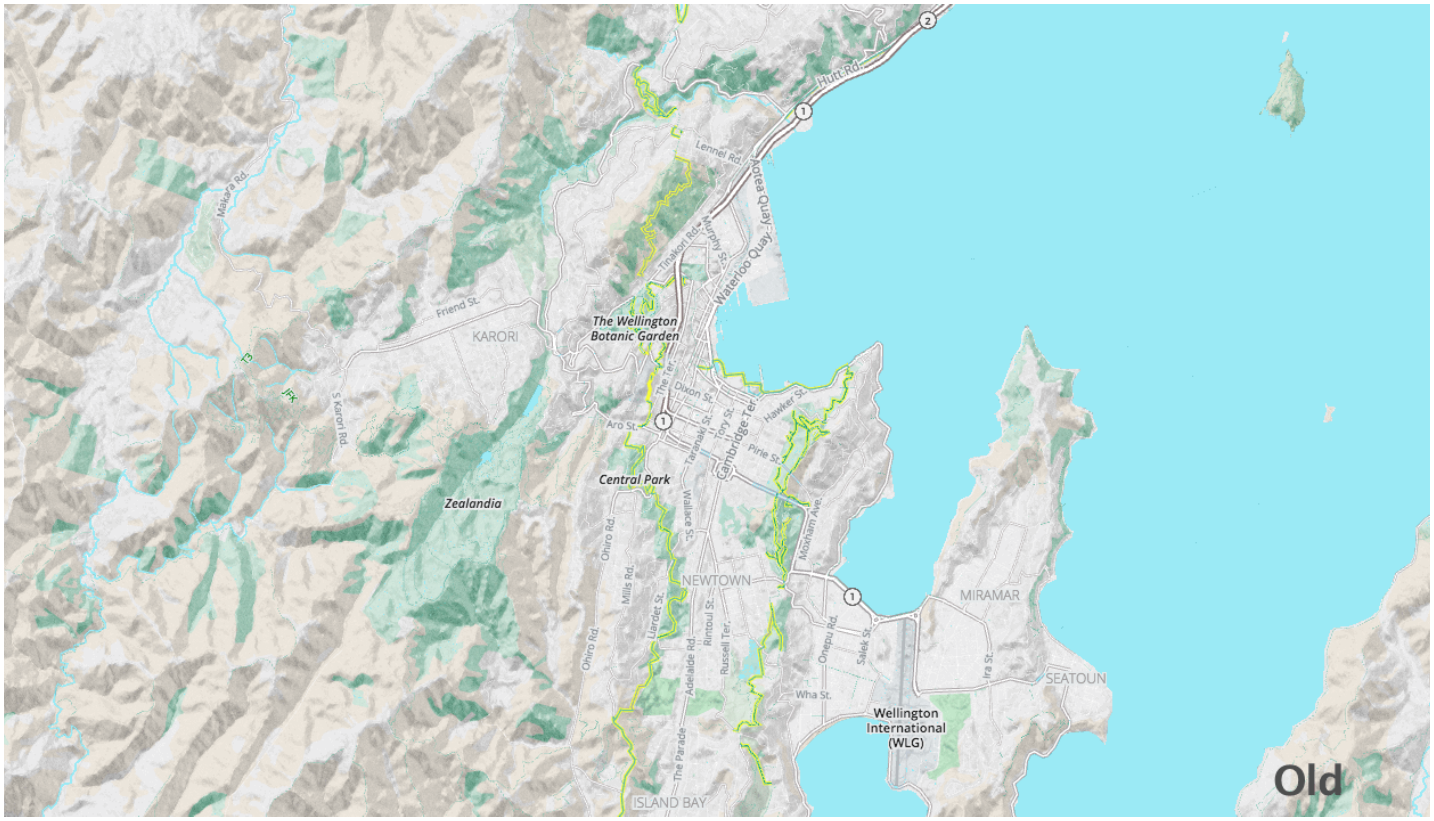
© 2017 Mapzen

Terrain Tiles v1.1 Update

terrain (/tag/terrain) data (/tag/data)

Just over a year ago, we **announced the Terrain Tiles service (https://mapzen.com/blog/terrain-tile-service/)** in cooperation with the Amazon Web Services (AWS) team. The Terrain Tiles Public Dataset offers digital elevation models (DEMs) for the entire Earth, with varying resolutions of data sourced from open data providers from around the world. More than 50 million terrain requests are made every day, making it one of the most popular Public Datasets offered by AWS.

The Mapzen Tiles team, with the help of **Seth Fitzsimmons (https://github.com/mojodna)**, have spent the last couple months making improvements to the Terrain Tiles in two important ways: We've added several newer, higher resolution sources in more countries, especially in Europe, and improved the software to use more modern Amazon Web Services products so that it can make more efficient use of cloud resources as we update the data.


Old
KARORI
The Wellington Botanic Garden
Central Park
Zealandia
NEWTOWN
MIRAMAR
SEATOUN
Wellington International (WLG)
ISLAND BAY

Updated terrain data around ***Wellington, New Zealand (https://tangrams.github.io/walkabout-style/#13/-41.29361/174.77926)***

We **added high resolution data (https://mapzen.com/documentation/terrain-tiles/data-sources/)** that covers the Arctic (including Alaska, Canada, Greenland, Norway, Iceland, and Russia), Australia, Austria, Canada, Europe (covering Albania, Austria, Belgium, Bosnia and Herzegovina, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Kosovo, Latvia, Liechtenstein, Lithuania, Luxembourg, Malta, Montenegro, Netherlands, Norway, Poland, Portugal, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, and United Kingdom), Mexico, New Zealand, Norway, and the United Kingdom.

This new coverage comes from governments releasing data they collect as open, accessible data. This allows us to include elevation data at resolutions of better than 5 meters per pixel in some parts of the world. This high resolution data is blended with other sources to offer the best elevation data we can find for any particular part of the Earth.

Updated terrain data near ***Lake Taymyr, Russia (https://tangrams.github.io/walkabout-style/#10/74.9007/104.2548)***

We also updated our software to **take advantage of modern cloud technology offered by Amazon Web Services (https://mapzen.com/blog/terrain-tiles-on-aws-batch/)**. The new system uses AWS Batch, Lambda, and S3 to allow practically unlimited scalability, letting us pre-generate the Terrain Tile dataset for the entire globe in under a week. The software, called **Marblecutter (https://github.com/mojodna/marblecutter/)**, also runs inside a Docker container, letting us prototype and test locally using the exact versions of software that were run in the cloud. Our system uses components from the existing aerial imagery project **OpenAerialMap (https://openaerialmap.org/)**, extending it to support single band DEM images. The updates and improvements to this software are also being used by OpenAerialMap to improve multi-band support. (We'll discuss this in depth in an upcoming blog post.)

In addition to updating the software and data, we also added support for 512 pixel tiles and buffered tiles at 260 and 516 pixels to **the Mapzen Terrain Tiles service (https://mapzen.com/documentation/terrain-tiles/)**. The 512 pixel tiles are useful for higher resolution screens and more efficient rendering of slippy maps while the 516 and 260 pixel tiles are helpful to support more advanced usage in 3D modeling tools or game engines.

If you're already using Mapzen Terrain Tiles, there's nothing you need to change. The Mapzen endpoints remain the same:

```
https://tile.mapzen.com/mapzen/terrain/v1/terrarium/{z}/{x}/{y}.png?api_key=mapzen-xxxxxx
https://tile.mapzen.com/mapzen/terrain/v1/normal/{z}/{x}/{y}.png?api_key=mapzen-xxxxxxx
https://tile.mapzen.com/mapzen/terrain/v1/geotiff/{z}/{x}/{y}.tif?api_key=mapzen-xxxxxxx
https://tile.mapzen.com/mapzen/terrain/v1/skadi/{N|S}{y}/{N|S}{y}{E|W}{x}.hgt.gz?api_key=
```

You can access the updated Terrain Tiles via the same Amazon S3 endpoints:

```
elevation-tiles-prod/terrarium/{z}/{x}/{y}.png
elevation-tiles-prod/normal/{z}/{x}/{y}.png
elevation-tiles-prod/geotiff/{z}/{x}/{y}.tif
elevation-tiles-prod/skadi/{N|S}{y}/{N|S}{y}{E|W}{x}.hgt.gz
```

You can get more information about each format in **Mapzen documentation (https://mapzen.com/documentation/terrain-tiles/formats/)** and the **Amazon Public Datasets page (https://aws.amazon.com/public-datasets/terrain/)**.

The updated tiles are already being used for hillshading in the **Walkabout basemap style (https://tangrams.github.io/walkabout-style)**. Check them out! (If you see any data problems, please **file an issue (https://github.com/tilezen/joerd/issues)**).

· 04 December 2017 ·

Ian Dees

Ian is on the tiles team, building squares of map data. Ian ❤️maps and bleeds nodes, ways, and relations.

Seth Fitzsimmons

Seth is a freelance systems architect living in Bend, Oregon. When not working on open data and open source projects, he enjoys spending time riding bikes, kayaking, and skiing in the greater Pacific Northwest.

© 2017 Mapzen

Using AWS Batch to Generate Mapzen Terrain Tiles

terrain (/tag/terrain) data (/tag/data)

The **Terrain Tiles dataset (https://mapzen.com/blog/terrain-tiles-v1.1)** consists of thousands of digital elevation model (DEM) data sources at various resolutions from around the globe, merged together, split into billions of map “tiles”, and stored on AWS’s S3 storage system. We developed a highly scalable system on top of AWS Lambda, Batch, and S3 in order to convert the sources, composite/merge them together, and split the merged files into tiles as quickly as possible.

Simplifying dependencies with containers

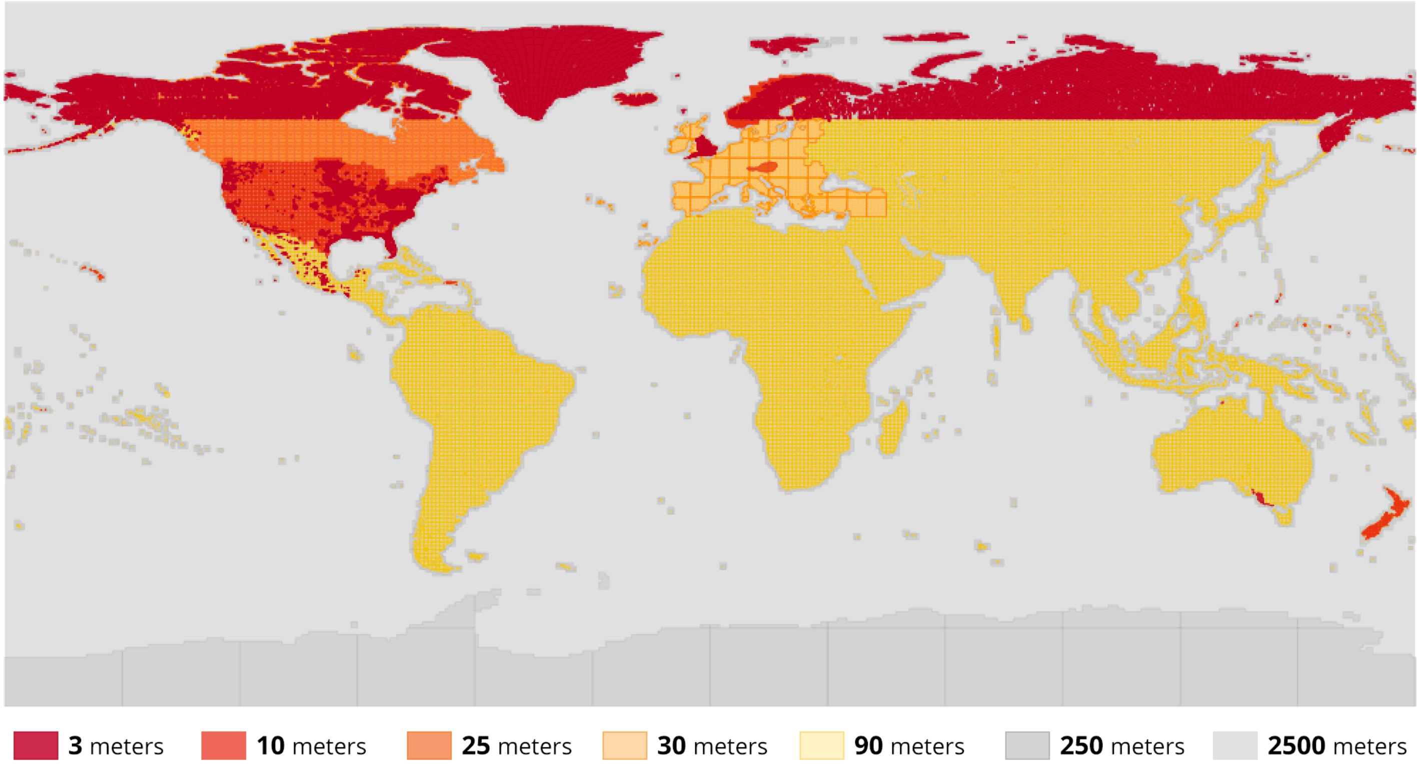
Building the Terrain Tiles dataset requires very recent versions of the popular geospatial software library GDAL. Since it’s relatively difficult to get the required version of GDAL installed, we built Dockerfiles to help us build with a consistent combination of operating system and dependencies. The **Dockerfiles we created (https://github.com/mojodna/marblecutter-tilezen/blob/master/Dockerfile)** include the correct versions of GDAL and the Python software that controls it to perform the various tasks required to build the tiles.

Generating cloud-optimized GeoTIFFs

The raw data comes from a wide range of sources from all around the world in different formats. The first step in the Terrain Tiles process is to convert these images into **Cloud Optimized GeoTIFFs (https://trac.osgeo.org/gdal/wiki/CloudOptimizedGeoTIFF)** and store them on S3 so our compositing toolchain can interact with them efficiently. **Cloud-optimized GeoTIFFs (https://medium.com/planet-stories/cloud-native-geospatial-part-2-the-cloud-optimized-geotiff-6b3f15c696ed)** store lower-resolution overviews in predictable locations within the same file as the original GeoTIFF, allowing us to utilize **GDAL’s built-in support (https://erouault.blogspot.fr/2017/10/gdal-and-cloud-storage.html)** for HTTP range requests.

To perform the transcoding step, we submit AWS Batch jobs that run **the transcode script (https://github.com/mojodna/marblecutter/blob/tilezen/bin/transcode.sh)** across the provider’s data. The transcoder script supports direct downloads from the provider, but in the case of a particularly complicated access methodology, we download the provider's data to our S3 bucket first and transcode from there. The transcoder script runs GDAL commands and writes the output cloud-optimized GeoTIFFs to a different S3 bucket.

Finally, the transcode script looks for areas in the image that should be transparent and builds a polygonal footprint for the coverage of the data using **rasterio (https://mapbox.github.io/rasterio/)**. That polygon, along with metadata about the source image, goes into a data of footprints that will be used during the compositing process.



An overview rendering of the roughly 32,000 source footprints in the footprints database.

Generating composite tiles

Once the source data has been transcoded into the cloud-optimized GeoTIFF format and indexed into a database, we overlay those sources on top of each other to form the composite outputs that become the the Terrain Tiles output. Our compositing software, called **Marblecutter (https://github.com/mojodna/marblecutter/)**, takes a bounding box, looks for source data in the footprints database, and fetches the pieces of the source data it needs to build the composite image.

Since we are compositing from cloud-optimized GeoTIFFs, the compositing software (which uses GDAL) only has to make HTTP range requests for the specific data it needs rather than downloading the entire source image. This dramatically speeds up the rendering process, especially when running in a cloud provider like AWS Batch.

We built two systems that use the marblecutter compositing library: one that runs in AWS Lambda to generate tiles on the fly in response to HTTP requests, and another that runs with AWS Batch to generate bulk tiles to cover the entire world.

The AWS Lambda tiler uses the AWS API Gateway system to respond to HTTP requests and re-generate a composite tile every time. This is particularly useful for debugging changes to the footprints database, allowing us to tweak the ordering or precedence of source images to ensure the output tiles use the best possible source data. (If you do see any data issues, **please file an issue (https://github.com/tilezen/joerd/issues)**.)

Finally, the marblecutter library is used to recursively generate a “pyramid” of map tiles starting from a particular e.g. zoom 9 tile and rendering every tile all the way down to a maximum zoom of e.g. zoom 15. This pyramid renderer system is run in a multithreaded Python environment across hundreds of AWS Batch-controlled EC2 cloud instances. Each Batch job enqueues an internal job on the multithreading queue for each tile in the pyramid of tiles. Each tile is also rendered in three different formats, meaning that each zoom 9-15 Batch job we run renders more than 16,000 tiles.

As we scaled this process up, we ran into unforeseen bottlenecks. First, the footprints database quickly ran out of available connections and the geospatial queries overwhelmed the CPU. To solve that, we added a step at the beginning of the Batch process to make a local in-memory cache of the footprints relevant to the pyramid of tiles getting rendered. Second, issuing requests for Cloud-optimized GeoTIFFs directly to S3 was slowing down compositing so we added a CloudFront cache in front of S3 to speed up the composite process. Finally, the S3 API couldn’t handle the burst of high throughput PUT requests, so we worked with the S3 team to develop a method to support higher instantaneous throughput by prefixing our keys with an MD5 hash. Once we generated the tiles with an hash prefix, we copied the tiles into place without the MD5 hash.

Using this setup on AWS Batch, we are able to generate more than 3.75 million tiles per minute and render the entire world in less than a week! These pre-rendered tiles get stored in S3 and are ready to use by anyone through the **AWS Public Dataset (https://aws.amazon.com/public-datasets/terrain/)** or through **Mapzen’s Terrain Tiles API (https://mapzen.com/documentation/terrain-tiles/)**.

· 05 December 2017 ·

Ian Dees

Ian is on the tiles team, building squares of map data. Ian ❤️ maps and bleeds nodes, ways, and relations.

Seth Fitzsimmons

Seth is a freelance systems architect living in Bend, Oregon. When not working on open data and open source projects, he enjoys spending time riding bikes, kayaking, and skiing in the greater Pacific Northwest.

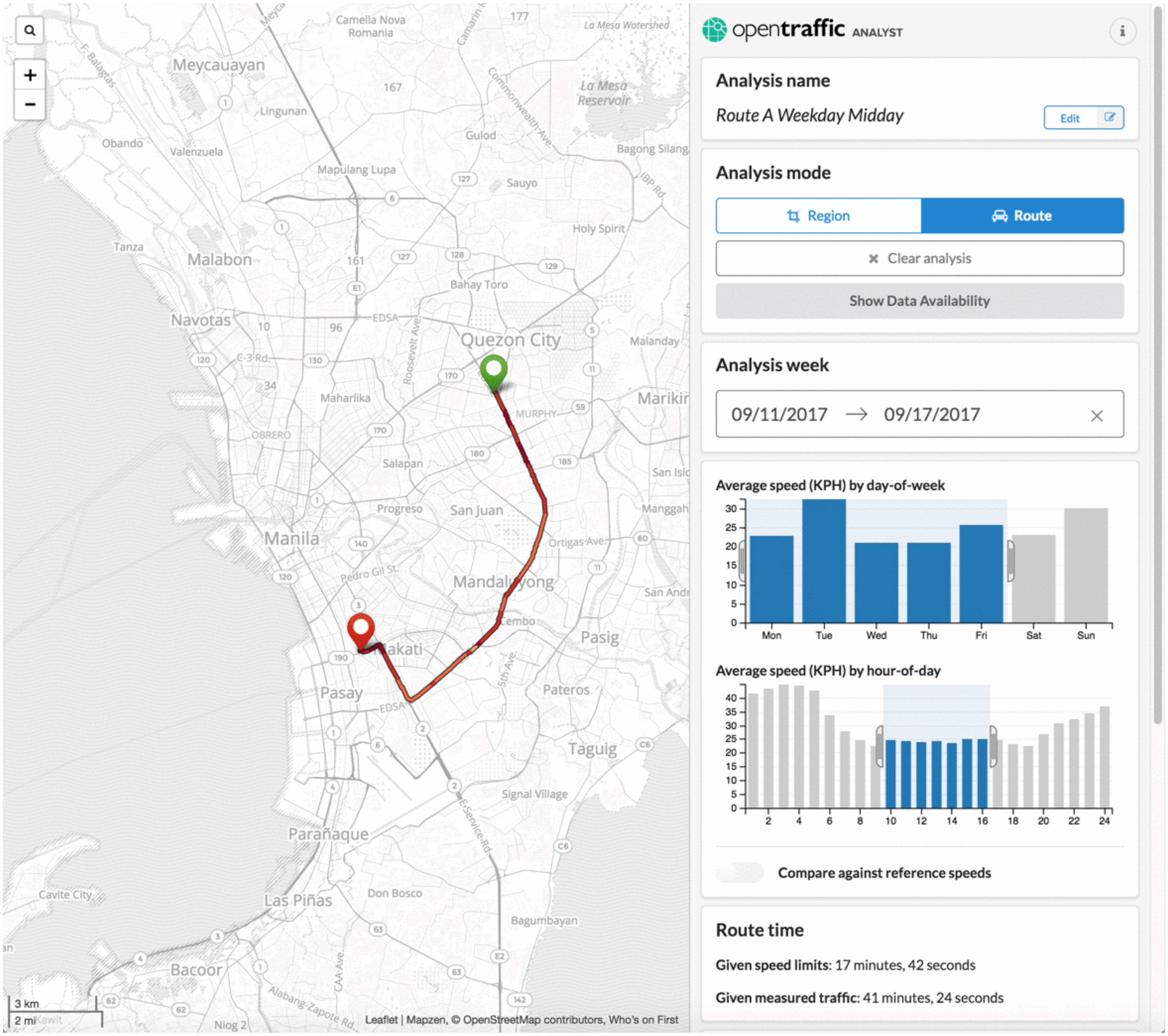
© 2017 Mapzen

Open Traffic platform released

mobility (/tag/mobility)

A year ago, Mapzen set out on **an adventure with the World Bank and their partners (http://www.worldbank.org/en/news/press-release/2016/12/19/the-world-bank-launches-new-open-transport-partnership-to-improve-transportation-through-open-data)** to create a global platform for turning anonymous GPS locations into roadway speeds, called Open Traffic. Now, we've finished our work on the platform and are pleased to hand Open Traffic to World Bank for operation.

Here are a few examples, from the Open Traffic Analyst User Interface, demonstrating what the platform provides:


opentraffic ANALYST
Edit
Region
Route
Clear analysis
Show Data Availability
Average speed (KPH) by day-of-week
0
5
10
15
20
25
30
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Average speed (KPH) by hour-of-day
0
5
10
15
20
25
30
35
40
2
4
6
8
10
12
14
16
18
20
22
24
Meycauayan
Malabon
Navotas
Quezon City
Manila
San Juan
Mandaluyong
Makati
Pasay
Pasig
Pateros
Taguig
Parañaque
Las Piñas
Bacoor
3 km
2 mi
Leaflet | Mapzen, © OpenStreetMap contributors, Who's on First

Analysis name

Route A Weekday Midday

Analysis mode

Route

Analysis week

09/11/2017 → 09/17/2017

Compare against reference speeds

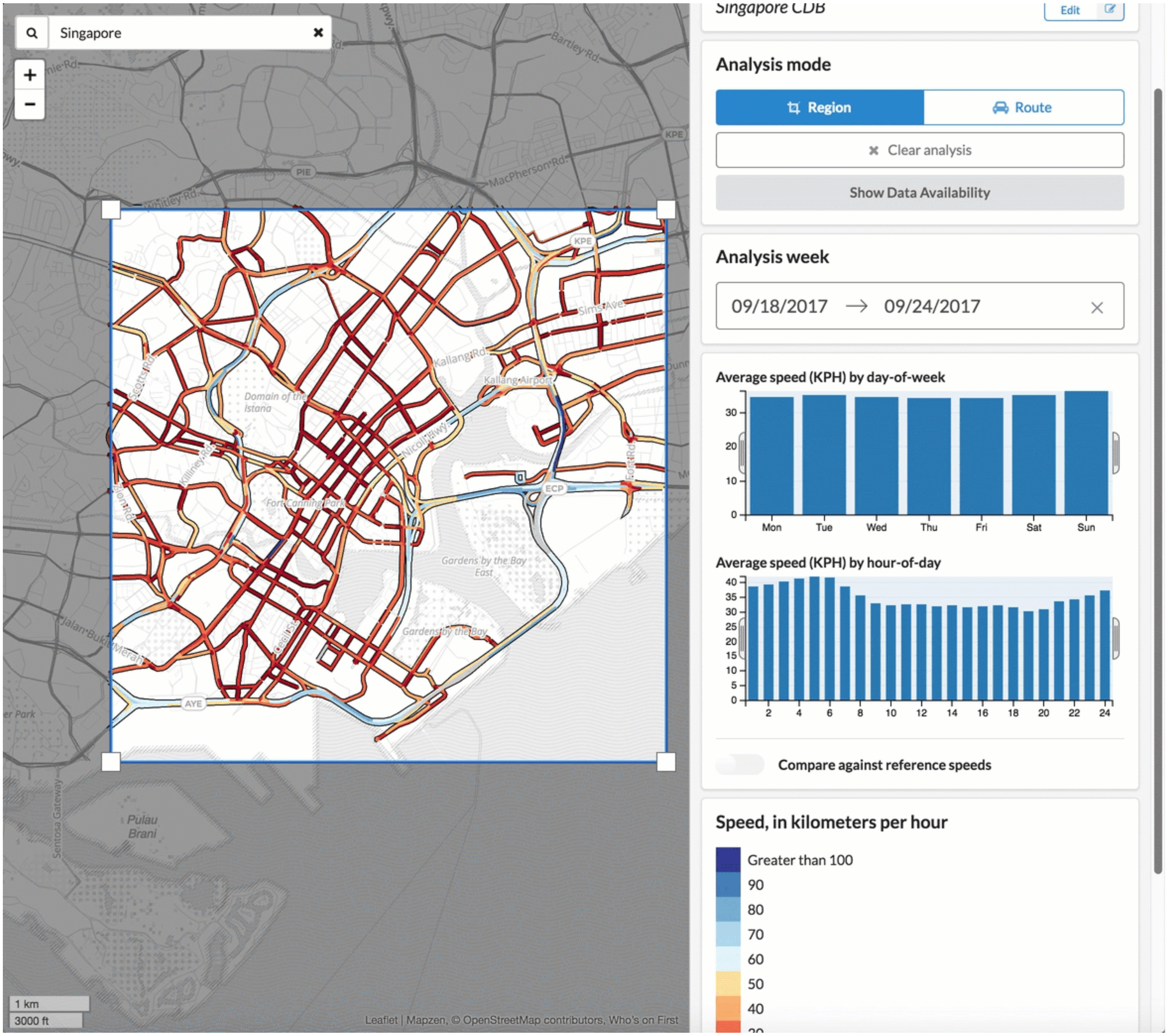
Route time

Given speed limits: 17 minutes, 42 seconds

Given measured traffic: 41 minutes, 24 seconds

Here's a route in Manila, Philippines (where we originally previewed (https://mapzen.github.io/open-traffic-poc-data-demo/) Open Traffic functionality). Look in the lower right hand of the Analyst UI to see how long this route is predicted to take at different times of day.

And here's how an entire region can be queried and visualized:

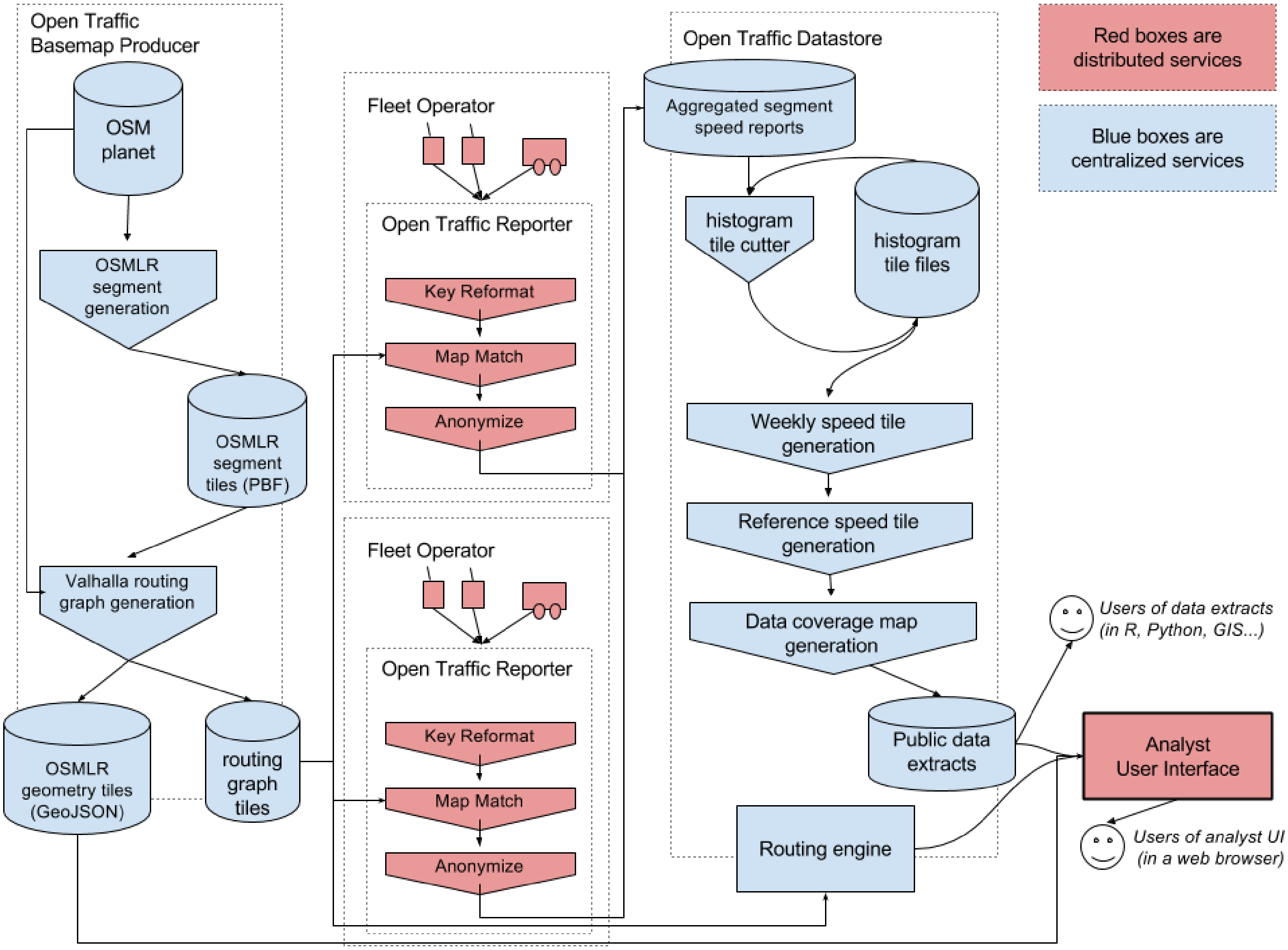

Singapore
+
−
Singapore CDB
Edit
Analysis mode
Region
Route
Clear analysis
Show Data Availability
Analysis week
Average speed (KPH) by day-of-week
0
10
20
30
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Average speed (KPH) by hour-of-day
0
5
10
15
20
25
30
35
40
2
4
6
8
10
12
14
16
18
20
22
24
Compare against reference speeds
Speed, in kilometers per hour
Greater than 100
90
80
70
60
50
40
PIE
KPE
ECP
AYE
MacPherson Rd
Bartley Rd
Sims Ave
Kallang Rd
Kallang Airport
Scotts Rd
Domain of the Istana
Killiney Rd
Fort Canning Park
Nicoll Hwy
Gardens by the Bay East
Gardens by the Bay
Jalan Bukit Merah
Pulau Brani
Sentosa Gateway
1 km
3000 ft
Leaflet | Mapzen, © OpenStreetMap contributors, Who's on First

Analysis mode: Region

Analysis week: 09/18/2017 → 09/24/2017

This is the central business district of Singapore, showing average speeds during a week in September, 2017.

We're especially proud of how this Analyst UI is powered only by static tile files. There is no API necessary behind the scenes. The Open Traffic platform includes many other technical innovations, to ensure data privacy and to scale to cover an entire planet's worth of motorized traffic on roadways. For a complete technical tour through the platform, see the **`opentraffic/otv2-platform` repository on GitHub (https://github.com/opentraffic/otv2-platform)**:


Open Traffic Basemap Producer
OSM planet
OSMLR segment generation
OSMLR segment tiles (PBF)
Valhalla routing graph generation
OSMLR geometry tiles (GeoJSON)
routing graph tiles
Fleet Operator
Open Traffic Reporter
Key Reformat
Map Match
Anonymize
Fleet Operator
Open Traffic Reporter
Key Reformat
Map Match
Anonymize
Open Traffic Datastore
Aggregated segment speed reports
histogram tile cutter
histogram tile files
Weekly speed tile generation
Reference speed tile generation
Data coverage map generation
Public data extracts
Routing engine
Red boxes are distributed services
Blue boxes are centralized services
Users of data extracts (in R, Python, GIS...)
Analyst User Interface
Users of analyst UI (in a web browser)

Code for all of the platform's components is on **GitHub (https://github.com/opentraffic)**, available under the LGPL license.

For more background on Mapzen's involvement in the Open Traffic project, here are our blog posts over the past year:

- **World Bank + Mapzen + partners = open traffic data (/blog/announcing-open-traffic/)** by Alyssa Wright and Drew Dara-Abrams
- **Open traffic data is the best traffic data (/blog/open-traffic-data/)** by Randy Meech
- **Open Traffic technical preview #1: OSMLR segments (/blog/open-traffic-osmlr-technical-preview/)** by Matt Amos, Kevin Kreiser, David Nesbitt, and Drew Dara-Abrams
- **OSMLR traffic segments for the entire planet (/blog/osmlr-2nd-technical-preview/)** by David Nesbitt and Drew Dara-Abrams
- **A "data-driven" approach to improving map-matching** by Max Gardner
    - **Part I: Validation (/blog/map-matching-validation/)**
    - **Part II: Parameter Optimization (/blog/data-driven-map-matching/)**
    - **Part III: Post-processing for Quality (/blog/map-matching-post-processing/)**

  - **Part IV: Map-matching and the Built Environment (/blog/map-matching-built-env/)**
- **OSMLR hits a "mile marker" (and joins AWS Public Datasets) (/blog/osmlr-released-as-public-dataset/)** by Drew Dara-Abrams and David Nesbitt

Many hands have been involved in making this effort a success. Beyond those listed on blog posts above: Grant Heffernan, Lou Huang, Greg Knisely, Kristen DiLuca, Mandy Kong, Duane Gearhart, John Oram, Patricio Gonzalez Vivo, and **many more at Mapzen (/about)**.

For more news on the Open Traffic platform, Please follow World Bank. Prospective partners are encouraged to write to Holly Krambeck **hkrambeck@worldbank.org (mailto:hkrambeck@worldbank.org)**.

We can't wait to see how the World Bank and its partners use the Open Traffic platform in efforts to lower greenhouse gas emissions, increase access to livelihoods, and improve quality of life in developing nations.

· 12 December 2017 ·

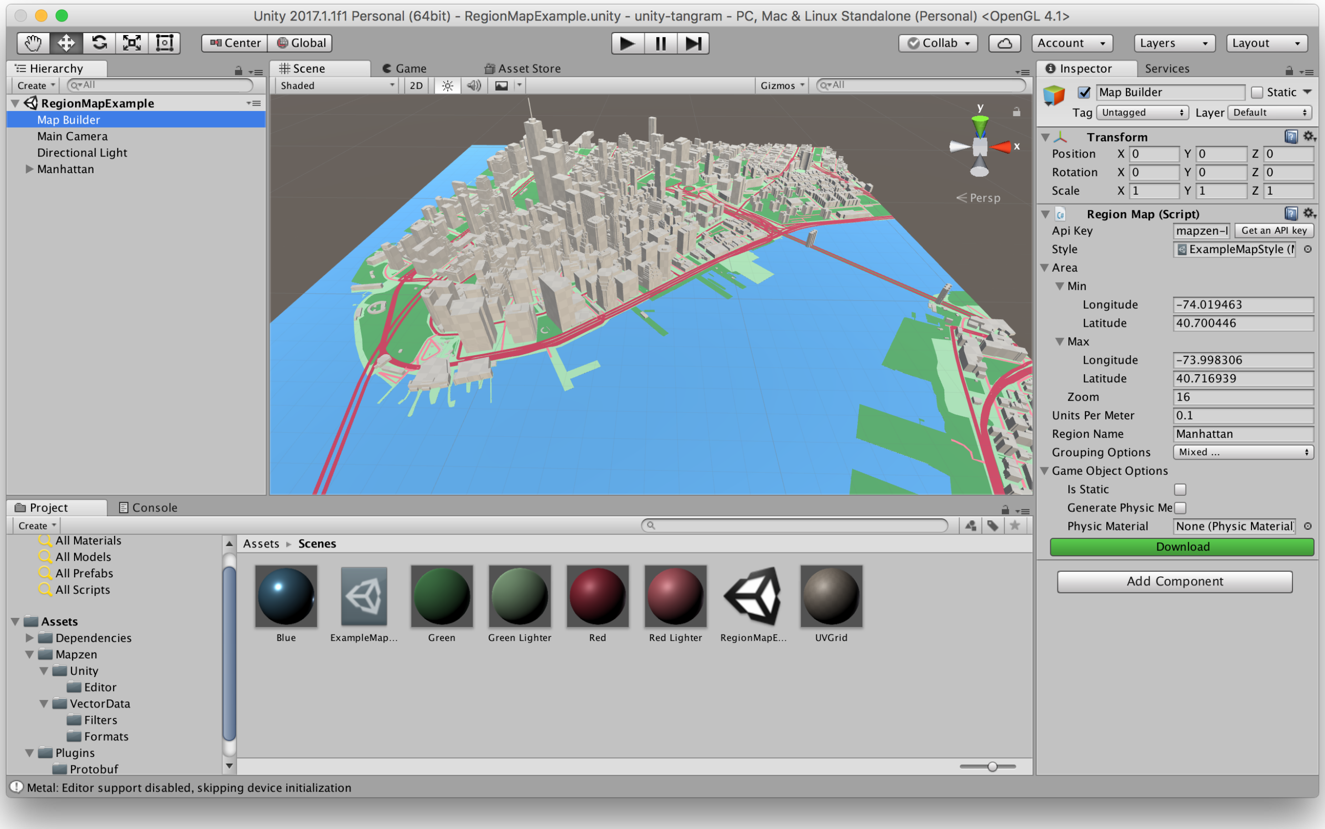
Drew Dara-Abrams

Drew leads Mapzen Mobility products (and aspires to being a flâneur).

© 2017 Mapzen

Announcing Tangram Unity

tangram (/tag/tangram)


Unity 2017.1.1f1 Personal (64bit) - RegionMapExample.unity - unity-tangram - PC, Mac & Linux Standalone (Personal) <OpenGL 4.1>

Map Builder ☑ ☐ Static
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Region Map (Script)
Api Key mapzen-I
Style ExampleMapStyle (I
Area
Min
Longitude -74.019463
Latitude 40.700446
Max
Longitude -73.998306
Latitude 40.716939
Zoom 16
Units Per Meter 0.1
Region Name Manhattan
Grouping Options Mixed ...
Game Object Options
Is Static ☐
Generate Physic Me ☐
Physic Material None (Physic Material)

Metal: Editor support disabled, skipping device initialization
Center
Global
Collab
Account
Layers
Layout
Hierarchy
Create
RegionMapExample
Map Builder
Main Camera
Directional Light
Manhattan
Scene
Game
Asset Store
Shaded
2D
Gizmos
Persp
Inspector
Services
Get an API key
Download
Add Component
Project
Console
All Materials
All Models
All Prefabs
All Scripts
Assets
Dependencies
Mapzen
Unity
Editor
VectorData
Filters
Formats
Plugins
Protobuf
Assets ▸ Scenes
Blue
ExampleMap...
Green
Green Lighter
Red
Red Lighter
RegionMapE...
UVGrid

We're excited to share that Mapzen is building open-source tools to use map data in Unity3D! You can try out **Tangram Unity (https://github.com/tangrams/tangram-unity)** using Mapzen's vector tile service and contribute your feedback and improvements.

Between Tangram and Tangram ES, you can already run Tangram maps on almost every current platform, but to date, Unity has been elusive. Games have a very different set of priorities when it comes to using maps.

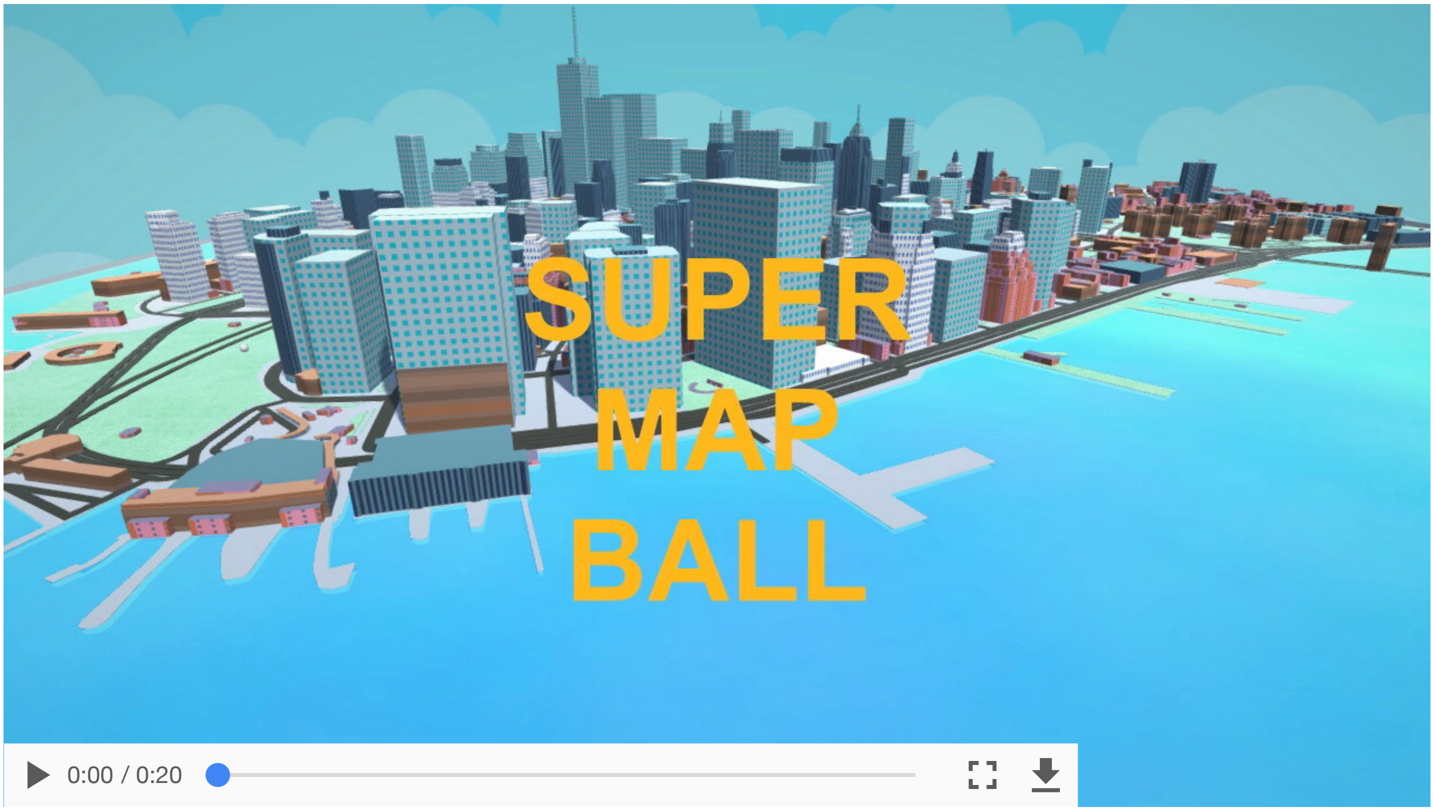
For instance, a mapping application needs to cover scales ranging from a city block to the entire world, but a game environment usually only needs a fixed scale. Running within the Unity engine also imposes some new engineering constraints. Our current architecture relies on multithreading and access to low-level graphics APIs, both of which have limited availability in Unity.

To meet these challenges, we decided to start from scratch. Tangram Unity brings what we've learned from Tangram and Tangram ES to the Unity ecosystem.

Tangram Unity is a library of code and GUI tools for building a map using real-world data in your Unity game scenes. Choose an area of data to download, configure a set of filtering and styling options, and then press a big green button to build an entire region of GameObjects with meshes. You can do all of this from the Unity Editor using GUI tools, or hook up your own pipeline using the classes in the library.

For now Tangram Unity is primarily for building map data into your scenes in the editor, before you deploy your game. This lets you view and edit the scene before the game is running, so you can make sure it looks exactly right for your game. Looking forward, we'd also like to enable building map data at run-time so you can procedurally generate a scene for any location in the world. We've also experimented with elevation data and plan to incorporate a terrain option in the future.

There's plenty you can do with the features in Tangram Unity right now. Our team put together a small game made in Unity both to help us develop the library and to play around with what's possible using OpenStreetMap data. Naturally, the game is open-source so you can go and check out **Super Map Ball (https://github.com/tangrams/super-map-ball)** for yourself!



This project is still young and growing! Want to try it out? Have ideas for other things the toolkit can do? **Head to the Github repository (https://github.com/tangrams/tangram-unity)** to see examples, file issues, or submit your own changes.

· 14 December 2017 ·

Matt Blair

Graphics engineer and perennial contrarian, contains strong feelings about video games.

© 2017 Mapzen

Whos On First Updates, 2017

whosonfirst (/tag/whosonfirst) data (/tag/data)

(https://mapzen-assets.s3.amazonaws.com/images/wof-updates/eraser.jpg)

So, what's new in Who's On First?

Before the year's end, I wanted to take a few moments to describe three issues that we've recently tackled in Who's On First. While many of the data issues in Who's On First can be complex, each issue has to be addressed with a unique set of tools, a specific workflow, and should follow simple rules that we've outlined to ensure our changes and adjustments can be added to the database.

We've tackled many large-scale issues in 2017, from the **Statoids and Mesoshapes imports (https://mapzen.com/blog/introducting-statoids/)**, to **updating neighbourhood records (https://mapzen.com/blog/neighbourhood-updates-two/)**, to the **GeoNames name localization (https://mapzen.com/blog/summer-2017-wof/)** work. I wanted to outline a few of the smaller issues we've dealt with in the past few months, and describe what it took to fix the issues.

The issues are below, each with a brief description of the problem and solution. Take a look and enjoy!

Santa Barbara Neighbourhoods

In October, **a user reported (https://github.com/whosonfirst-data/whosonfirst-data/issues/997)** that the record for the neighbourhood of Midco in California had **a *large* geometry (https://github.com/whosonfirst-data/whosonfirst-data/blob/028f288cc2a2995edb819a90d395a73c553017b0/data/420/552/455/420552455.geojson)**. So large, in fact, the geometry spanned some 50+ miles, from Santa Maria, CA to Santa Barbara, CA. This geometry was causing the **Who's On First API (https://mapzen.com/blog/whosonfirst-api/)** to return peculiar hierarchy results for this area.

This geometry issue was two-fold; first, we needed to shrink the size of the Midco neighbourhood geometry down to a size that actually represents Midco and secondly, we needed to find a source for new neighbourhood records in Santa Barbara (Midco covered the entire city of Santa Barbara).

Once Midco was **cut down to size (https://whosonfirst.mapzen.com/spelunker/id/420552455/)**, **openly licensed (https://github.com/whosonfirst/whosonfirst-sources/blob/master/sources/santabar.json)** neighbourhood data on **Santa Barbara's website (https://maps.santabarbaraca.gov/Html5Viewer/Index.html?configBase=/Geocortex/Essentials/REST/sites/City_of_Santa_Barbara__Public/viewers/SantaBarbaraPublic/virtualdirectory/Resources/Config/Default)** was found. After downloading the neighbourhood data, I was able to mint new `wof:id` values (thank you, **Brooklyn Integers (www.brooklynintegers.com)** for new Who's On First records), map the source properties into applicable Who's On First **properties (https://github.com/whosonfirst/whosonfirst-properties/tree/master/properties)**, and import them as new records!

To download Santa Barbara neighbourhoods, use our Bundler tool, **here (https://whosonfirst.mapzen.com/spelunker/download/85921779/?exclude=nullisland)** or take a look at the geometries below!

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Above: New neighbourhood records for Santa Barbara, CA.

Australia Regions

While the type of geometry and property updates in Australia are not new to Who’s On First, they represent the type of updates that many Who’s On First records have received in the past and what many records will receive in the future. In a **pull request (https://github.com/whosonfirst-data/whosonfirst-data/pull/688)** in the `whosonfirst-data` repository, new geometries were added to the records for regions in Australia.

To do this, the source data from **Australia Open Data (https://github.com/whosonfirst/whosonfirst-sources/blob/master/sources/austriaod.json)** was loaded into **QGIS**

(https://www.qgis.org/en/site/). A new property field was added to the source data with the corresponding `wof:id` for each feature; this is essentially allows us to “link” the new source data features to the existing Who’s On First records. Then, using a simple Python script, the geometries for each Who’s On First record were updated and existing geometries were saved as alt-geometry files.

And, just like that, Australia regions were updated!

An example of these geometry updates is shown below. Zooming out, you’ll see the older, less detailed geometry we maintained for Christmas Island. Zooming in, you’ll see the new, more detailed geometry - quite the improvement!

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Above: Comparison of region geometries for Christmas Island.

Leningrad and St Petersburg

Piggy-backing on the historical administrative work we started in **Yugoslavia (https://whosonfirst.mapzen.com/spelunker/id/1108955789/)**, we adjusted records in and around **St. Petersburg, Russia (https://whosonfirst.mapzen.com/spelunker/id/1158857531/)** to reflect the administrative changes in that area since the **Russian Revolution (https://en.wikipedia.org/wiki/Saint_Petersburg#Revolution_and_Soviet_Era_(1917%E2%80%931941))**.

An issue was filed (https://github.com/whosonfirst-data/whosonfirst-data/issues/948) in the `whosonfirst-data` repository, letting us know there were two distinct records in Who's On First for St. Petersburg - one for St. Petersburg and one for Leningrad. Upon further inspection, we actually had four duplicate records in that area, all of which were merged, superseded, and updated.

Take a look at the record for the St. Petersburg locality, **here (https://whosonfirst.mapzen.com/spelunker/id/1158857531/)**.

Wrapping Up

And, with that, I hope this is given you insight into the daily workings of geometry and property issues in Who's On First!

As always, if you would like to contribute to Who's On First, file a new issue in the **whosonfirst-data (https://github.com/whosonfirst-data/whosonfirst-data/issues/new)** repository or **drop us a line (mailto:hello@mapzen.com)**!

Photo Credit: ***Carol VanHook. (Flickr, CC BY 2.0) (https://flic.kr/p/mAYbSi)***

· 14 December 2017 ·

Stephen Epps

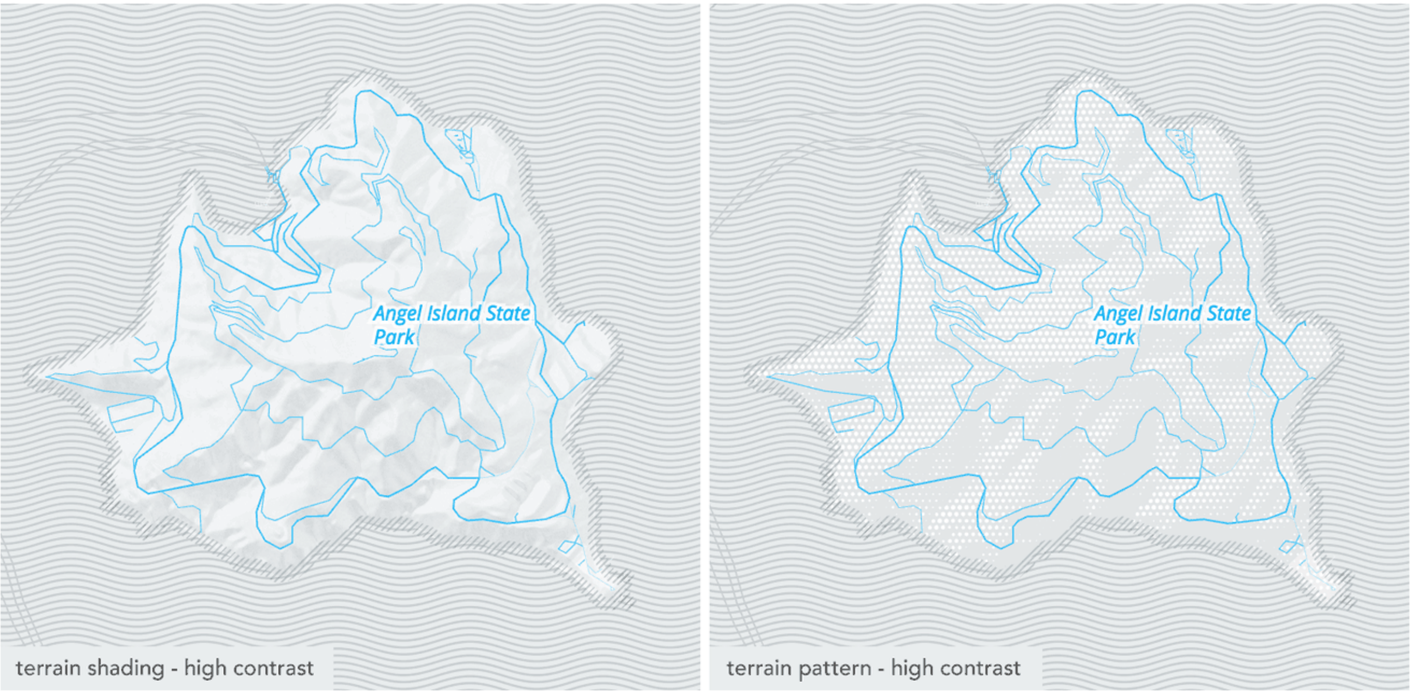
Stephen is Mapzen's gardener and fixer-upper of geographic data.

© 2017 Mapzen

Refill Terrain

cartography (/tag/cartography) tangram (/tag/tangram) terrain (/tag/terrain)

Introducing Refill with shaded relief! We have added two terrain styles for Refill: **terrain shading** and **terrain pattern**. The default terrain styles work well for general usage with a “blush” shading treatment and we also offer **high-contrast** (*-dark*) options that increases the relief shading, useful when your overlay data is related to elevation. See below:

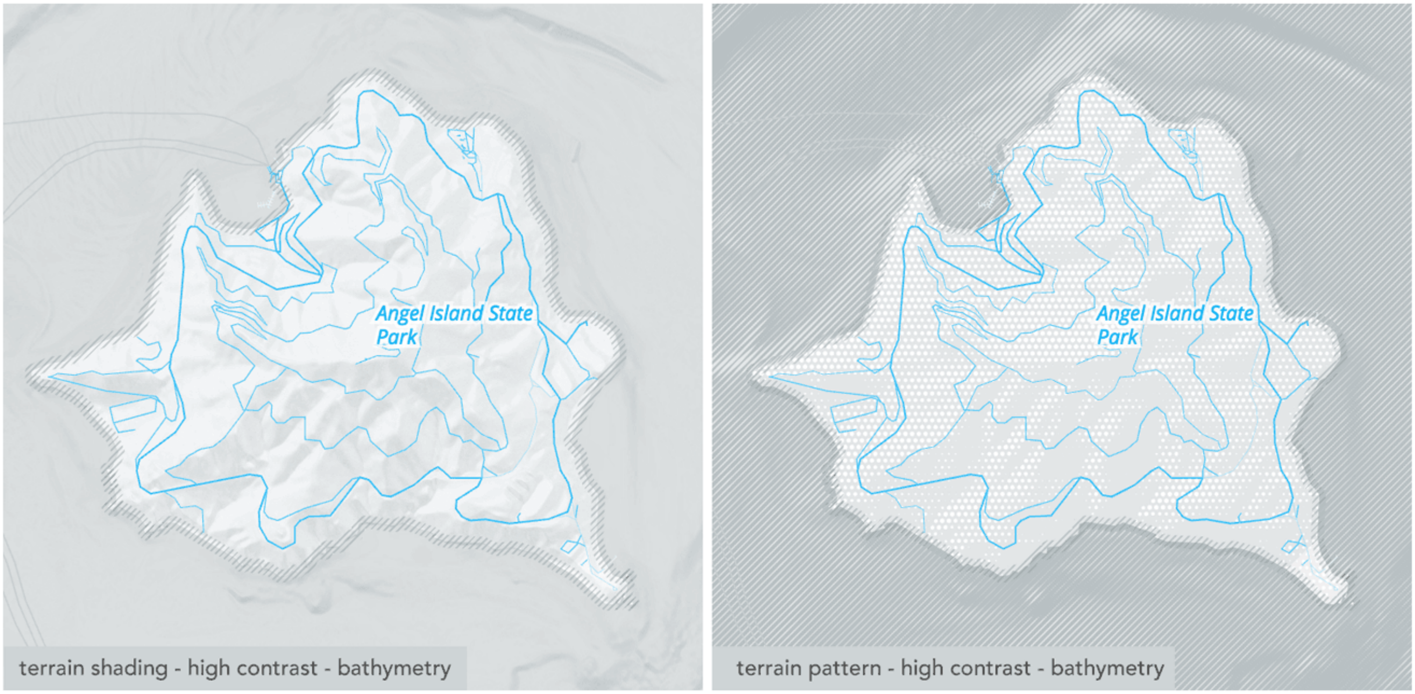

Angel Island State Park
terrain shading - high contrast
Angel Island State Park
terrain pattern - high contrast

With the **v1.1 terrain (https://mapzen.com/blog/terrain-tiles-v1.1/)** update earlier in December high resolution terrain data is available in more places in Refill and **Walkabout (https://mapzen.com/documentation/cartography/styles/#walkabout)**: the Arctic, Australia, Austria, Canada, Europe-wide, Mexico, New Zealand, Norway, and the United Kingdom.

Combined with the other **Refill API options (https://mapzen.com/documentation/cartography/api-reference/)** for theme colors, levels of detail, label steps, no-texture, and booleans for coastline, bathymetry, transit, and 3d buildings there are now more than 250,000 combinations available to remix! Support for multiple languages adds even more map variations, and fully realizes the power of vector tiles and client-side rendering. **Preview all Refill combos » (http://tangrams.github.io/refill-style/#15/37.7926/-122.4003)**

Bathymetry

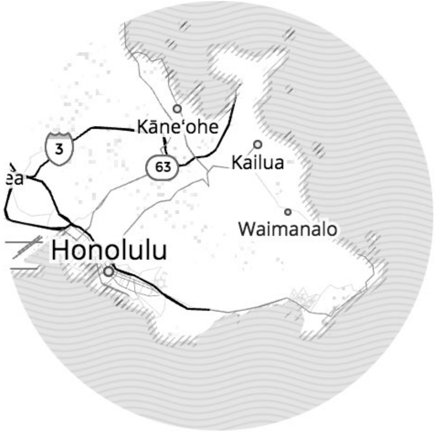
To see the terrain shape underwater in the oceans and select inland water bodies, turn on the new SDK bathymetry option. Bathymetry comes in two contrast variants as well, customized for each terrain theme.


Angel Island State Park
terrain shading - high contrast - bathymetry
Angel Island State Park
terrain pattern - high contrast - bathymetry

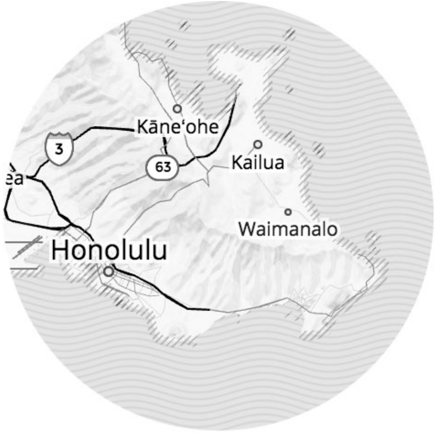
Colorful Terrain

Terrain is available with all color themes. See the variety of combinations for terrain on two of the light-colored themes below, Refill black and Refill blue.

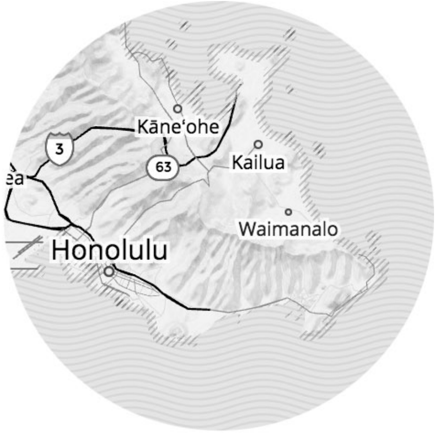
Refill Black


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

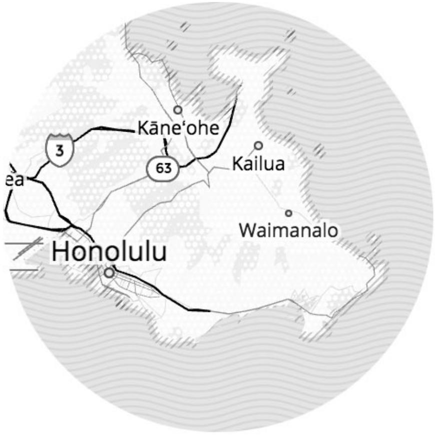
original/default


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

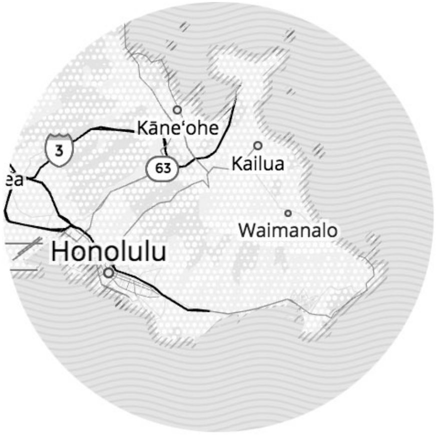
terrain shading


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

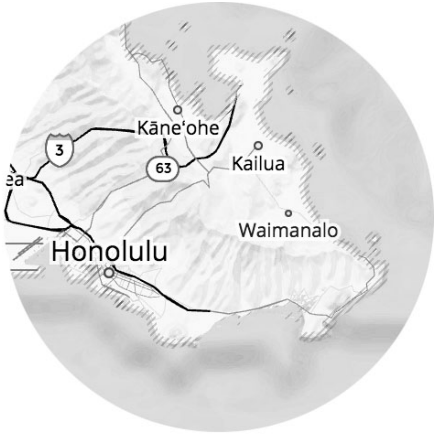
terrain shading - high contrast


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

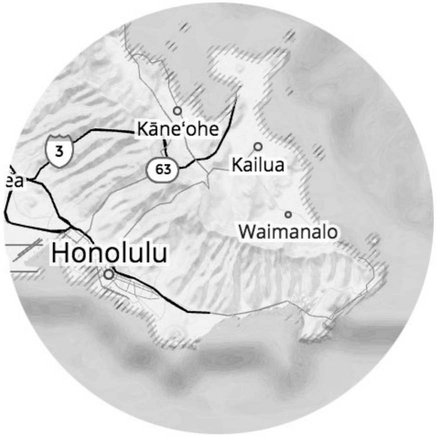
terrain pattern


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

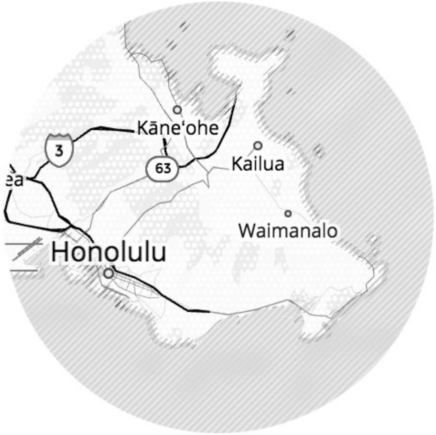
terrain pattern - high contrast


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

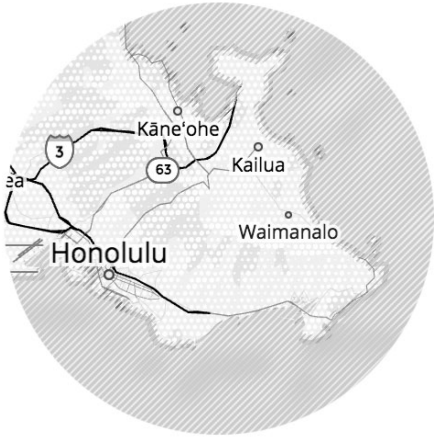
terrain shading - bathymetry


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

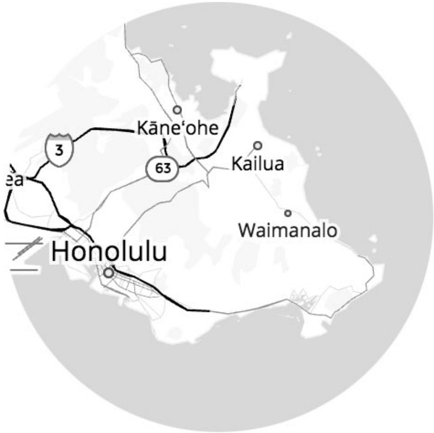
terrain shading - high contrast - bathymetry


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

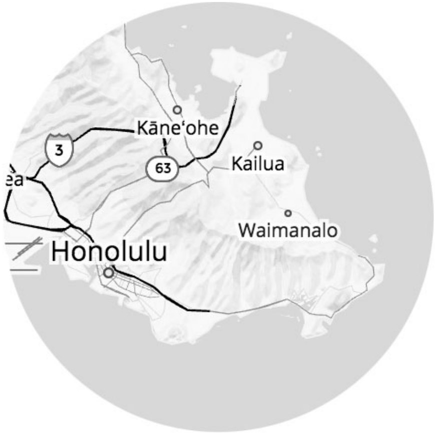
terrain pattern - bathymetry


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

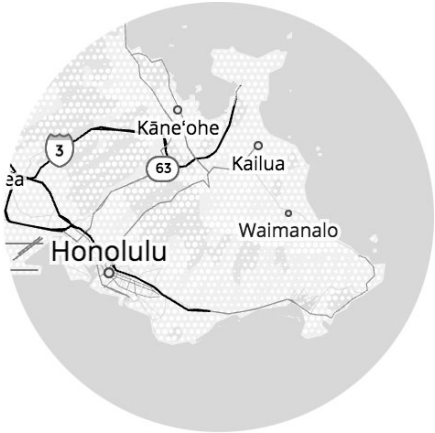
terrain pattern - high contrast - bathymetry


Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

no texture

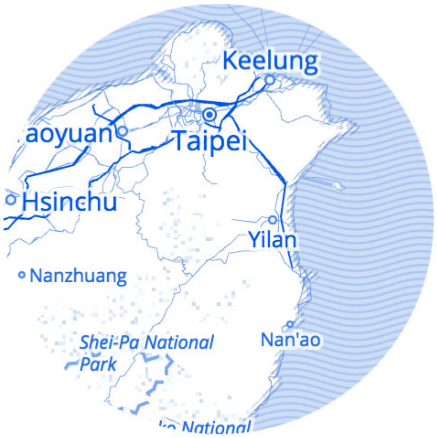

Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

terrain shading - no texture

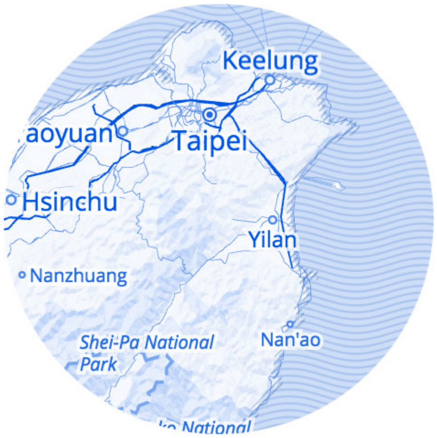

Kāneʻohe
3
63
Kailua
ea
Waimanalo
Honolulu

terrain pattern - high contrast - no texture

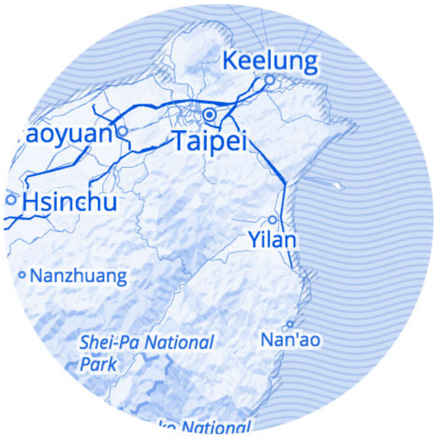
Refill Blue


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

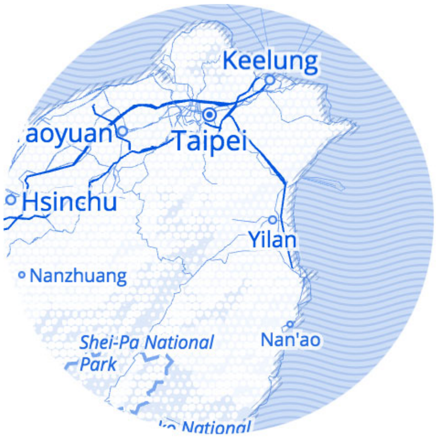
original/default


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

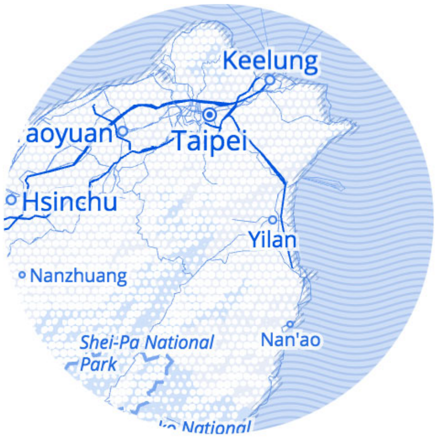
terrain shading


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

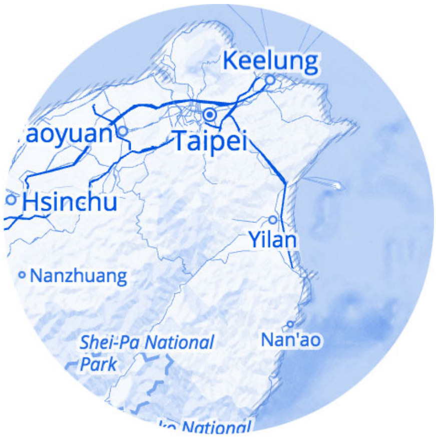
terrain shading - high contrast


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

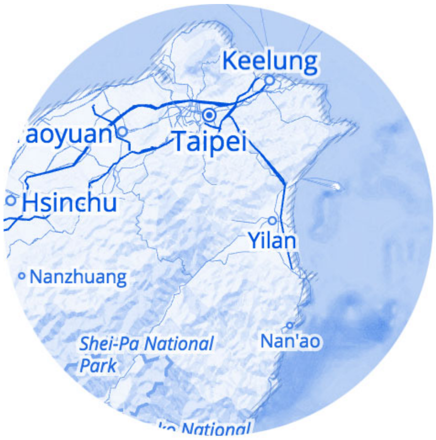
terrain pattern


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

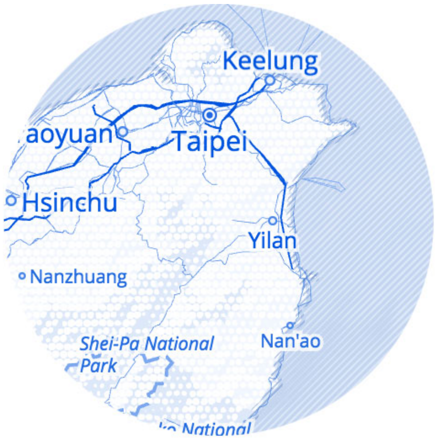
terrain pattern - high contrast


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

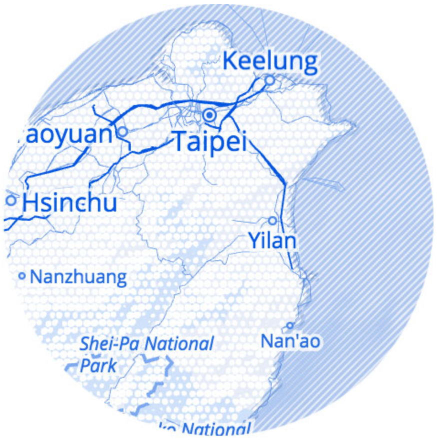
terrain shading - bathymetry


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

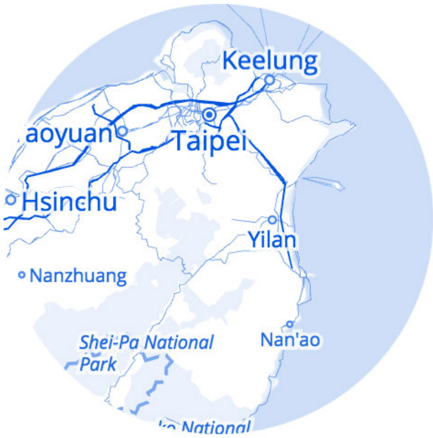
terrain shading - high contrast - bathymetry


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

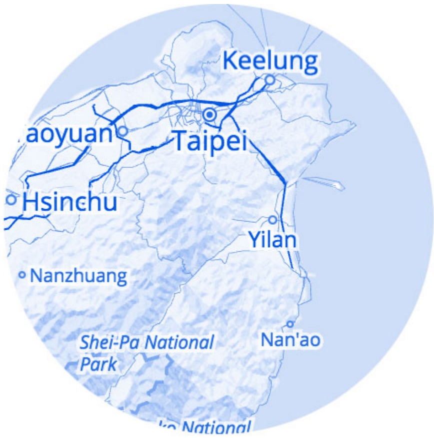
terrain pattern - bathymetry


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao

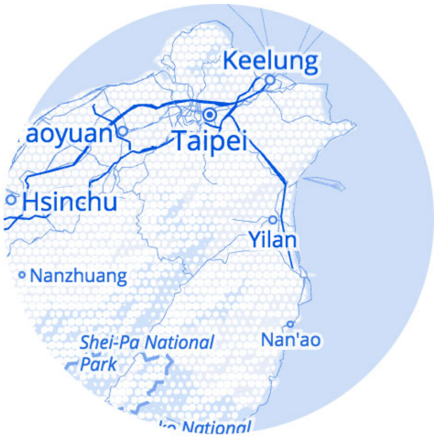
terrain pattern - high contrast - bathymetry


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao
National

no texture


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao
National

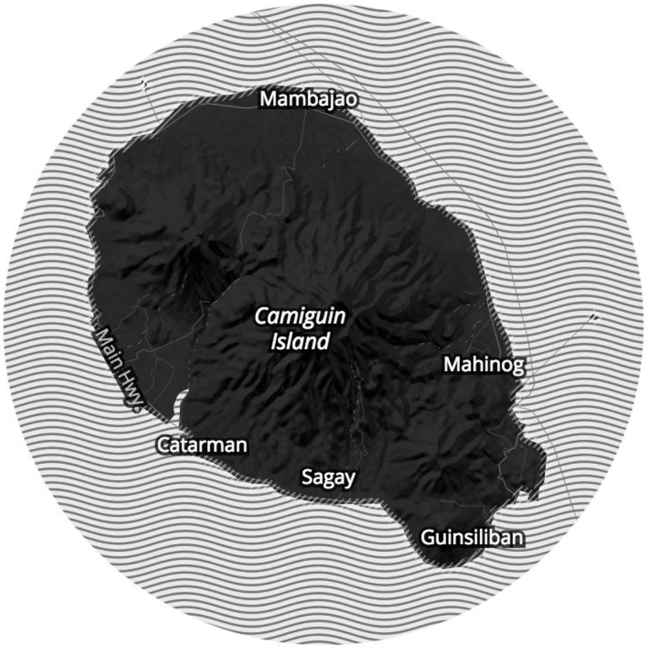
terrain shading - high contrast - no texture


Keelung
aoyuan
Taipei
Hsinchu
Yilan
Nanzhuang
Shei-Pa National Park
Nan'ao
National

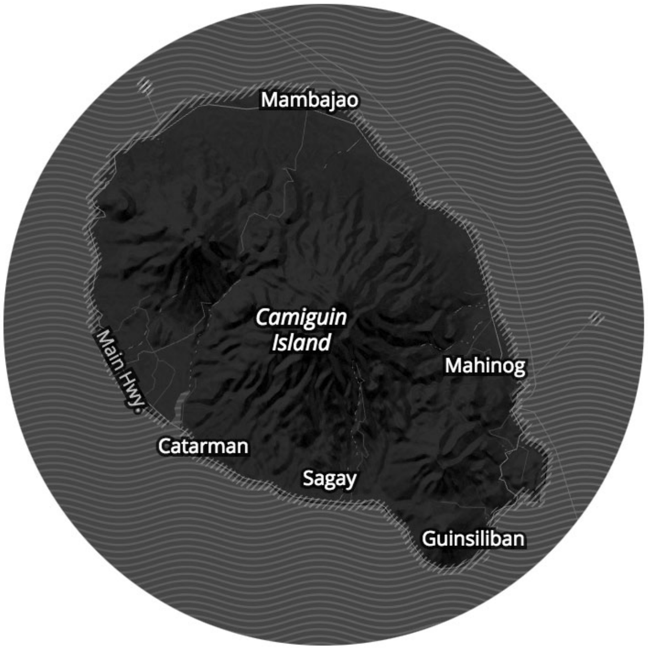
terrain pattern - high contrast - no texture

Dark Themes

We've added another color theme to our theme catalog! Inverted now has two options, with light and dark water areas. See the new **inverted-dark** option in John's **fire map (https://burritojustice.github.io/viirs-modis-fire-map/index.html#lat=34.4193&lng=-119.2569&z=10.9375)** showing the Thomas fire in southern California featuring live updates with latest available satellite observations.


Mambajao
Camiguin Island
Main Hwy.
Mahinog
Catarman
Sagay
Guinsiliban

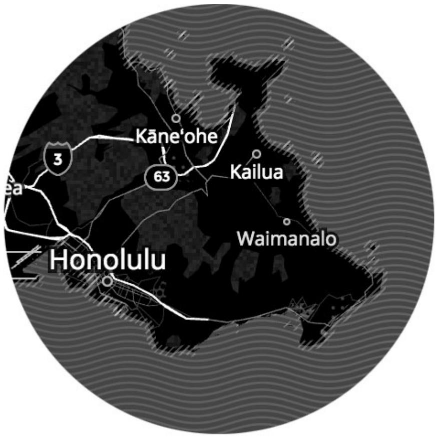
inverted


Mambajao
Camiguin Island
Main Hwy.
Mahinog
Catarman
Sagay
Guinsiliban

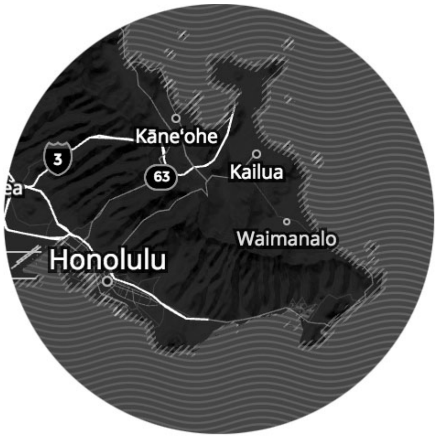
inverted dark

Refill Inverted Dark

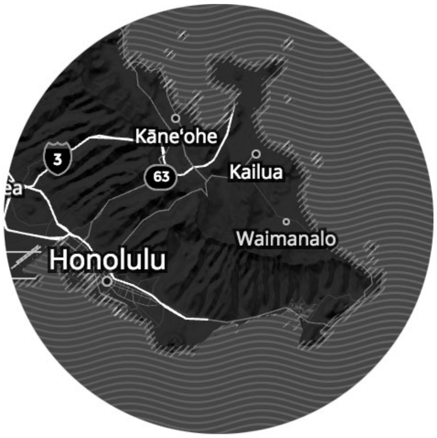
Here are the terrain combinations on the new inverted dark color theme.


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

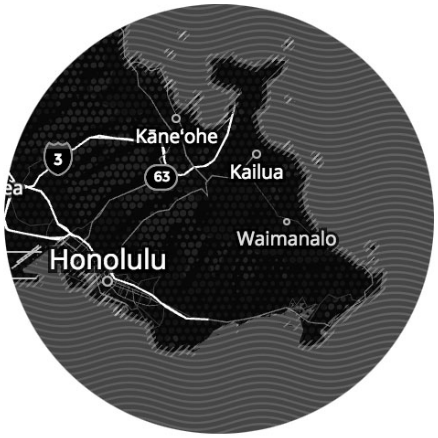
original/default


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

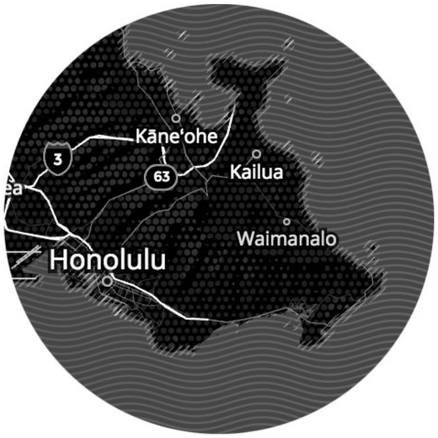
terrain shading


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

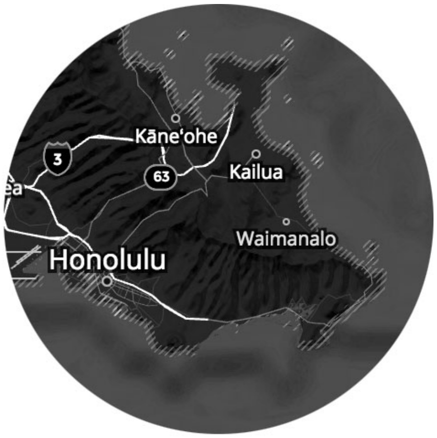
terrain shading - high contrast


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

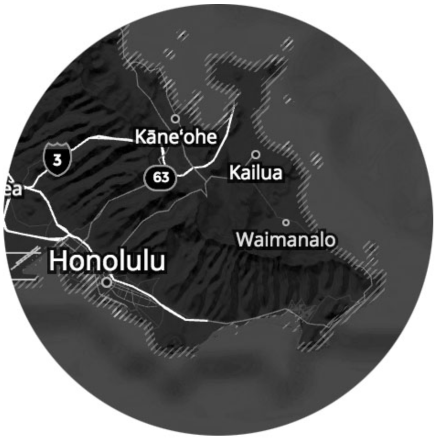
terrain pattern


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

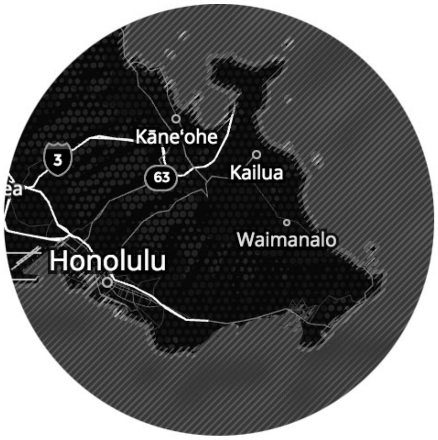
terrain pattern - high contrast


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

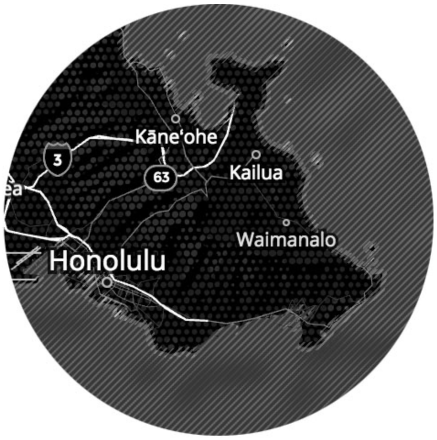
terrain shading - bathymetry


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

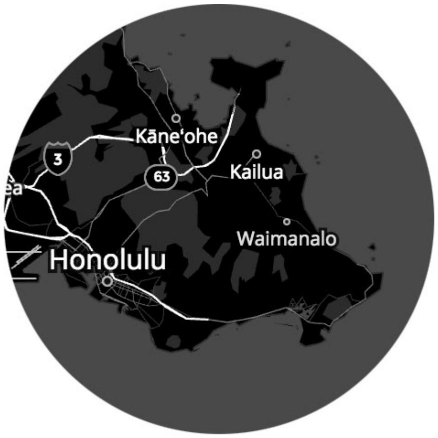
terrain shading - high contrast - bathymetry


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

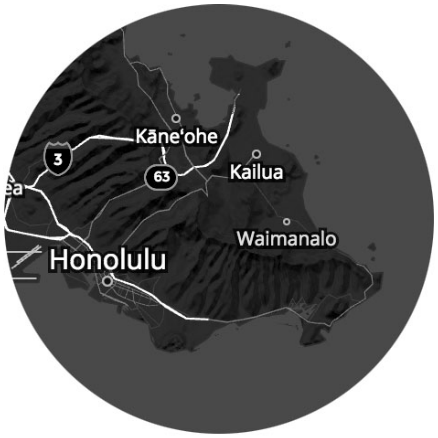
terrain pattern - bathymetry


Kāne'ohe
3
63
Kailua
ea
Waimanalo
Honolulu

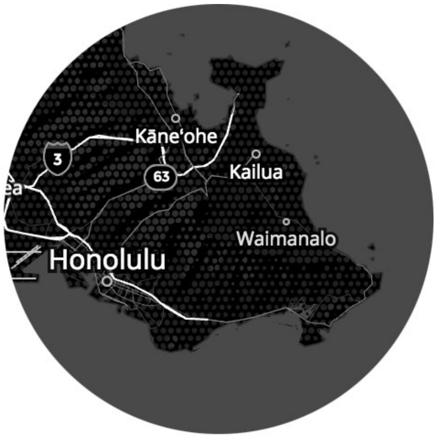
terrain pattern - high contrast - bathymetry


Kāne'ohe
3
63
Kailua
Waimanalo
Honolulu

no texture


Kāne'ohe
3
63
Kailua
Waimanalo
Honolulu

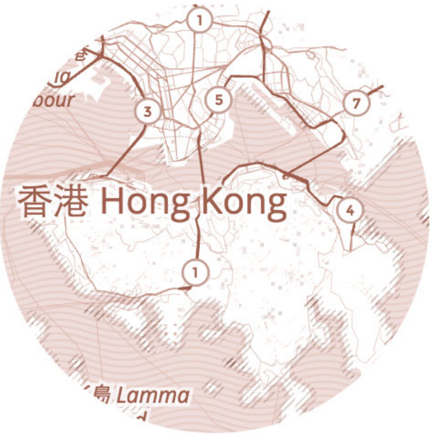
terrain shading - high contrast - no texture


Kāne'ohe
3
63
Kailua
Waimanalo
Honolulu

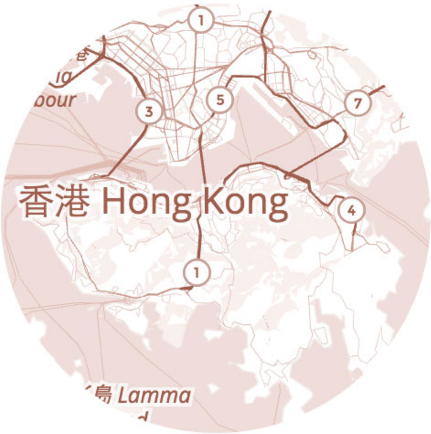
terrain pattern - high contrast - no texture

No Texture and Coastline Options

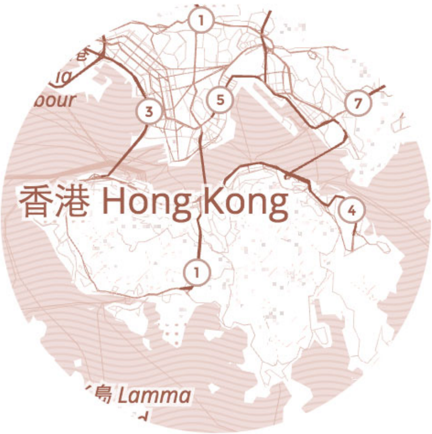
As seen from the Refill combination images above, Refill also has a **no-texture** option to remove all shader patterns. The no-texture option has flat areas of color instead of the variety of shader patterns. There is also a new theme option of keeping shader patterns on while removing the coastline shader only via a new API global. See Refill sepia with these options below.


香港 Hong Kong
Lamma
1
3
5
7
4

Refill sepia


香港 Hong Kong
Lamma
1
3
5
7
4

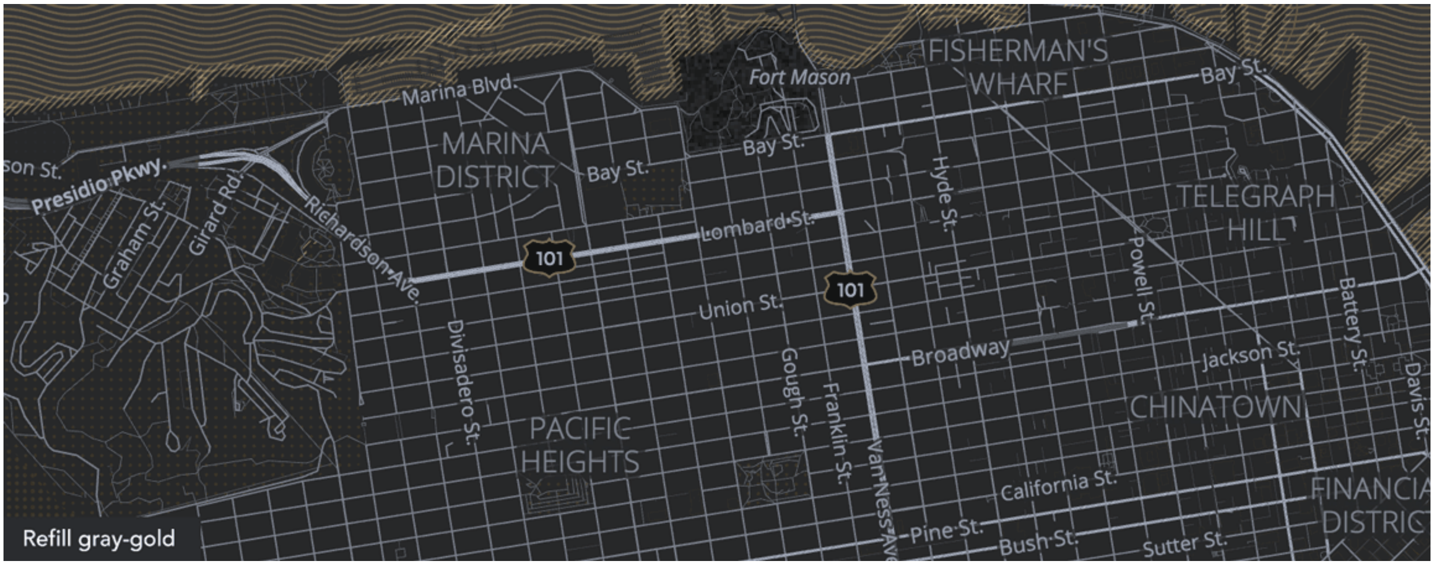
Refill sepia - no texture


香港 Hong Kong
Lamma
1
3
5
7
4

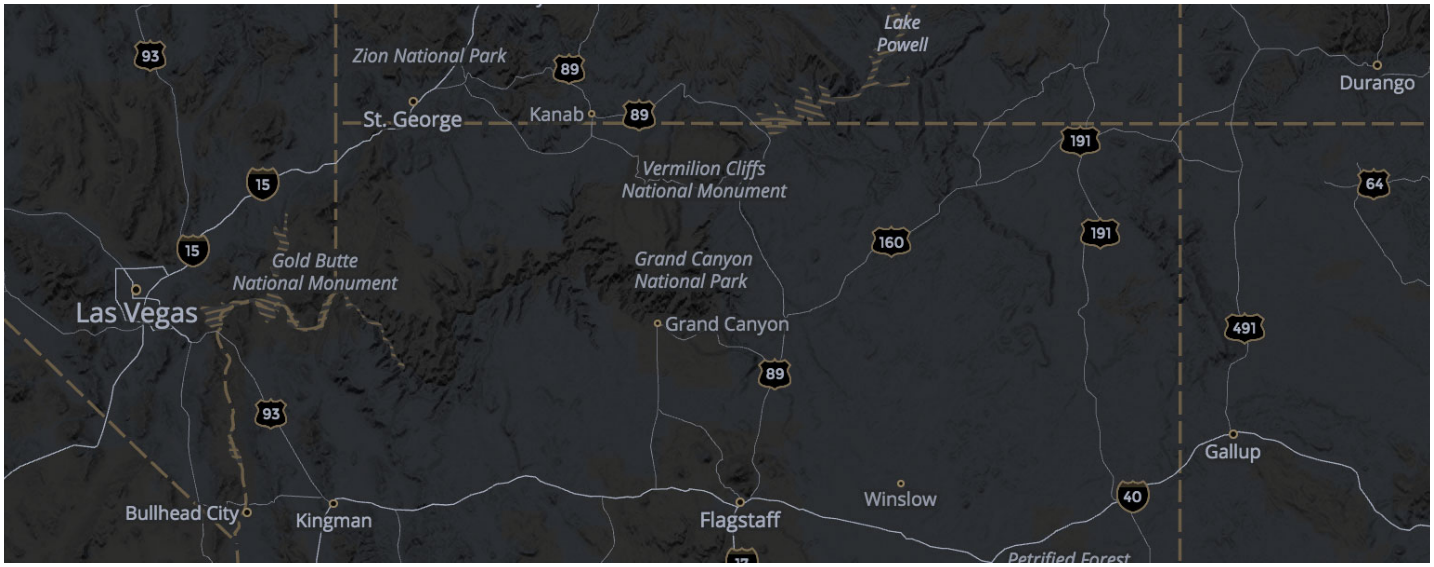
Refill sepia - coastline off

Overall Improvements

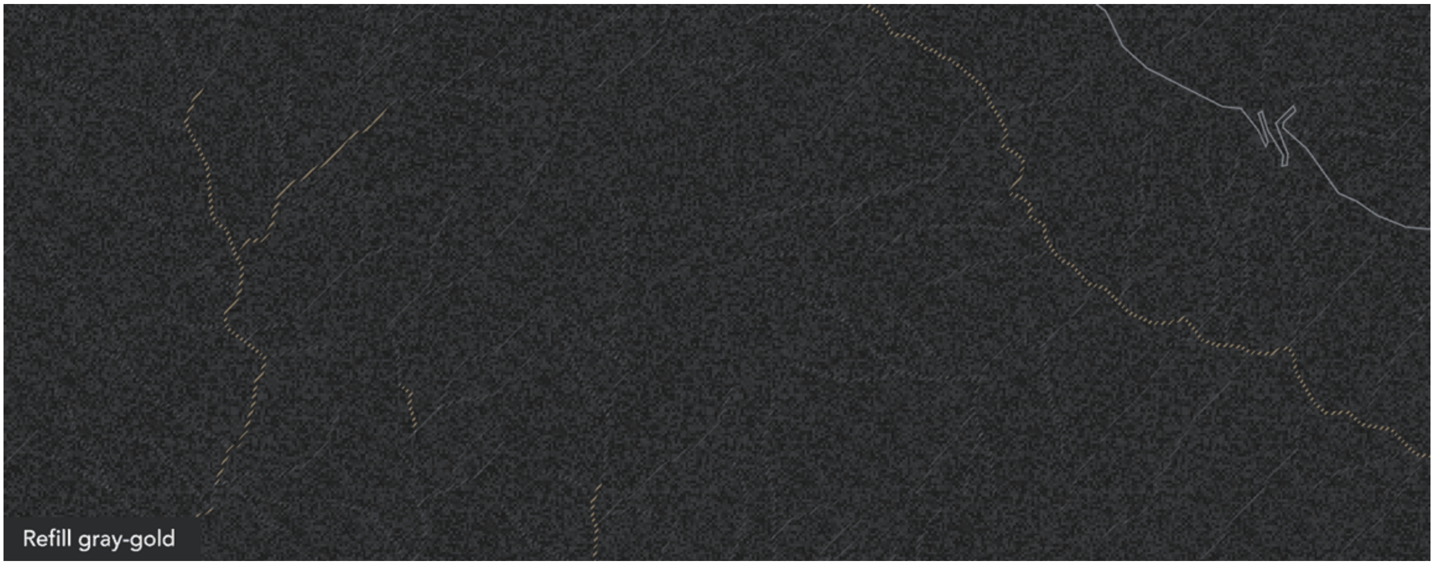
Several improvements have been made to improve the legibility of labels for all the variations of terrain and color themes. Labels have been adjusted to have a 75% halo opacity. See San Francisco in **gray-gold** below.


son St.
Marina Blvd.
Fort Mason
FISHERMAN'S WHARF
Bay St.
MARINA DISTRICT
Bay St.
Bay St.
Presidio Pkwy.
Graham St.
Girard Rd.
Richardson Ave.
101
Lombard St.
Hyde St.
TELEGRAPH HILL
Powell St.
101
Union St.
Battery St.
Broadway
Jackson St.
Divisadero St.
Gough St.
Franklin St.
Van Ness Ave
CHINATOWN
Davis St.
PACIFIC HEIGHTS
California St.
FINANCIA DISTRIC
Pine St.
Bush St.
Sutter St.
Refill gray-gold

Regional borders are now dashed beginning at zoom level 7.

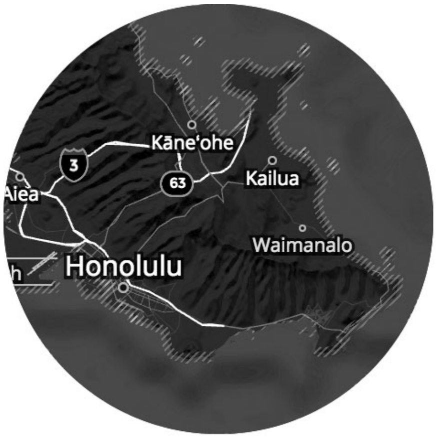

93
Zion National Park
89
Lake Powell
Durango
St. George
Kanab
89
191
Vermilion Cliffs National Monument
15
64
15
160
191
Gold Butte National Monument
Grand Canyon National Park
Las Vegas
Grand Canyon
491
89
93
Gallup
Winslow
40
Bullhead City
Kingman
Flagstaff
Petrified Forest

Rivers and streams colors and widths have been adjusted to work on all combinations of terrain. See rivers and streams in Shasta Lake area in California below.

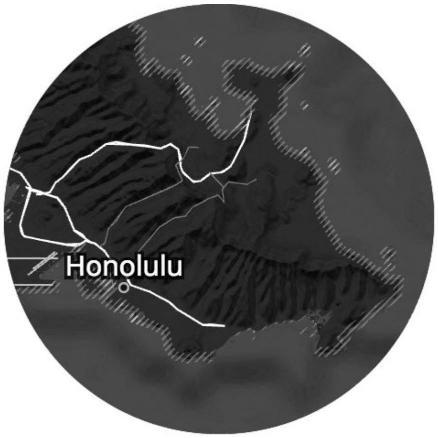

Refill gray-gold

Levels of Detail

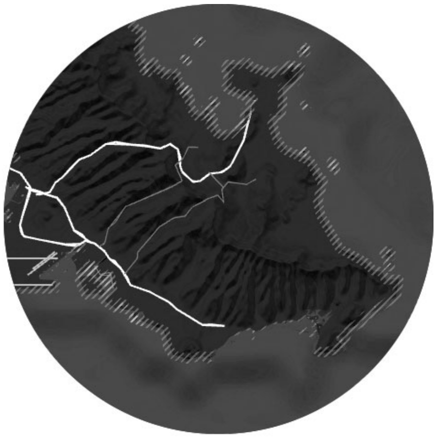
Play with combinations of label steps and levels of detail. Below are a few combinations on Refill **inverted-dark**.


Kāne'ohe
3
63
Kailua
Aiea
Waimanalo
Honolulu

label 5 - level of detail 10



label 2 - level of detail 5



label 0 - level of detail 6

Importing Terrain

The example below imports Refill, adds Blue as a color theme, shows more labels, and adds terrain shading. Bathymetry is also enabled via a global.

Order matters: import the terrain themes *after* all other themes.

```
import:
    - https://mapzen.com/carto/refill-style/10/refill-style.zip
    - https://mapzen.com/carto/refill-style/10/themes/color-blue.zip
    #- https://mapzen.com/carto/refill-style/10/themes/color-inverted-dark.zip
    - https://mapzen.com/carto/refill-style/10/themes/label-10.zip
    #- https://mapzen.com/carto/refill-style/10/themes/no-texture.zip
    - https://mapzen.com/carto/refill-style/10/themes/terrain-shading.zip
    #- https://mapzen.com/carto/refill-style/10/themes/terrain-shading-dark.zip
    #- https://mapzen.com/carto/refill-style/10/themes/terrain-pattern.zip
    #- https://mapzen.com/carto/refill-style/10/themes/terrain-pattern-dark.zip

global:
    sdk_bathymetry: true
#    sdk_coastline: false
```

Migrating to Refill terrain, inverted-dark, and no-texture

The following Mapzen basemap versions introduce **terrain** themes and related options:

- Refill: `10.0.0`

· 15 December 2017 ·

Geraldine Sarmiento

Geraldine is cartographer at Mapzen. She is addicted to shaders.

Nathaniel Vaughn Kelso

Map geek, cartographer, and data omnivore. Nathaniel leads the data team at Mapzen and vacations @nullisland.

© 2017 Mapzen

WOF in a Box (part 1)

whosonfirst (/tag/whosonfirst)

One of central design constraints of Who's On First is that we don't pick your database for you. We publish our gazetteer in **GeoJSON (https://whosonfirst.mapzen.com/data)**, CSV, and **SQLite (https://whosonfirst.mapzen.com/sqlite)** and you can decide how you want to import into your software stack of choice. We aspire to support all programming languages, on whatever platform you happen to be using.

We have written our own editorial tools and utility scripts in a variety of languages including **PHP (https://github.com/whosonfirst?utf8=%E2%9C%93&q=&type=&language=php)**, **Python (https://github.com/whosonfirst?utf8=%E2%9C%93&q=&type=&language=python)**, and **Go (https://github.com/whosonfirst?utf8=%E2%9C%93&q=&type=&language=go)**. Some of these are general purpose libraries, like **py-mapzen-whosonfirst (https://github.com/whosonfirst/py-mapzen-whosonfirst)**, and some are narrowly scoped utilities like **go-whosonfirst-pip-v2 (https://github.com/whosonfirst/go-whosonfirst-pip-v2)**.

Describing the *full range* of the tools and services that compose Who's On First, in its entirety, is beyond the scope of this guide, but it's a good start. So far we've relied on a mix of shell scripts, Makefiles, and the AWS-hosted Chef deployment system known as OpsWorks. This exercise in documenting the installation of the WOF stack onto "bare metal" is a way to ensure the stack can exist outside the AWS ecosystem.

This guide is the first in a two part series describing how to set up tools for managing Who's On First records on a single Ubuntu Linux server:

- **WOF Spelunker (https://github.com/whosonfirst/whosonfirst-www-spelunker/)** for exploring WOF records through a **web interface (https://whosonfirst.mapzen.com/spelunker)**
- **Who's On First API (https://github.com/whosonfirst/whosonfirst-www-api/)** (aka **Mapzen Places (https://places.mapzen.com/)**) to query the data via a **REST API (https://mapzen.com/documentation/places)**
- **Boundary Issues (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/)** for editing records using a bespoke web-based CMS

Part two will cover the last pair of web applications, this first part focuses on the Spelunker and the various dependencies required to get it up and running.

We'll begin by setting up a fresh server from scratch, installing all the dependencies required to host **administrative data (https://github.com/whosonfirst-data/whosonfirst-data)** and each of the tools. Some of the underlying technologies we'll be using include:

- Python
- Elasticsearch
- nginx
- Apache
- PHP
- MySQL

The Apache/PHP/MySQL parts will come in the second part of the guide.

One thing that *isn't* part of the install process yet is Docker. We have started to Dockerize the setup process, but we have a long way to go before the whole stack is covered. If anyone out there is inclined to help us translate our current setup process into Docker, we would definitely appreciate the help.

These docs are a work in progress. If you have thoughts or concerns, or if you get stuck, we would appreciate your feedback! Another caveat is that we are in the midst of migrating our servers from Ubuntu 14.04 to 16.04 (and PHP from version 5 to 7), so there may be some hiccups along the way.

The server itself

This guide assumes you have, in your possession, an actual physical computer to install all of this onto. It could be the computer you're reading this on, or a brand new one. The steps are not tied to a particular kind of hosting, and you can certainly set everything up in a cloud-hosted server if you prefer.

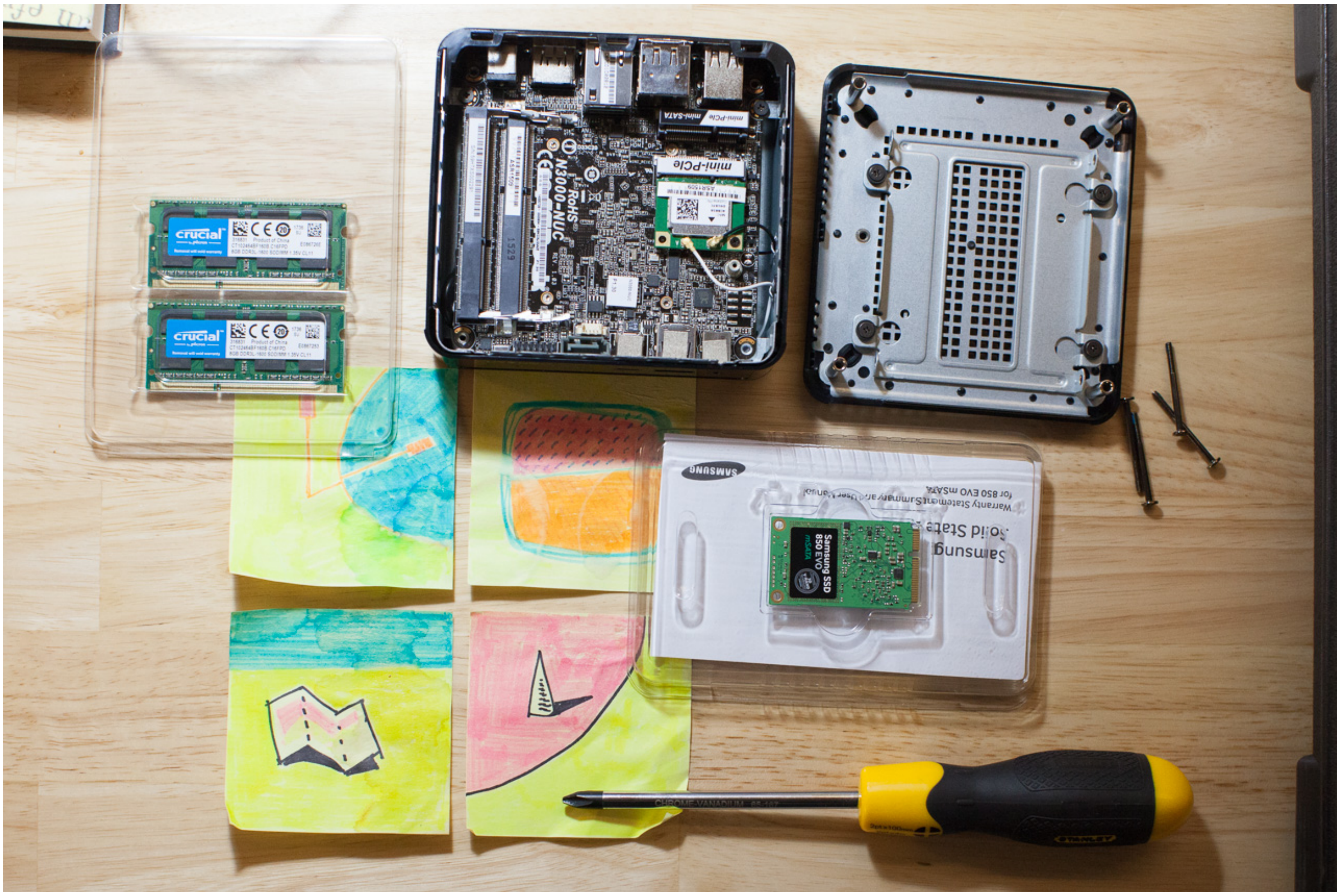
We will be using **Ubuntu 16.04 LTS (http://releases.ubuntu.com/16.04/)**. Things should work on other versions and flavors of Linux, but you may need to adjust some of the setup commands. You can cherry pick various parts of this guide that are interesting to you if you don't feel like setting up the full stack. Each service is designed to work independently of the others and they plug into each other by editing configurable URLs. There are also feature flags to enable/disable functionality, so if you see anything missing it may just be turned off.

I recommend running your WOF in a Box on an **Intel NUC (https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html)**-based server. It's more powerful than a Raspberry Pi, but is affordable and easy to source. Unlike rack-mounted hardware, it will sit quietly on your desk and you can transport it easily in a backpack. The base systems run in the $100-200 range, and you decide how much RAM and disk to install on it. The total system should cost $300-700 depending on your chosen specs. I recommend maxing out the RAM, if you can afford it, but disk space is easy to expand later with external storage.

Hardware checklist:

- Intel NUC
- RAM, the more the better (usually two sticks of equal size)
- mSATA SSD hard drive
- A USB wired keyboard
- An HDMI-capable monitor
- Screwdrivers (to install RAM, SSD disk)
- An Ethernet cable & a network you can plug it into
- A USB memory stick to install Ubuntu

I am not going to describe the hardware setup process in detail here, but if you have all the right parts on hand it should be pretty clear what plugs in where when you open up the case.



Operating System

Start by installing Ubuntu on the machine. If you're going with a cloud-hosted server option, like AWS or DigitalOcean, you can skip this part.

1. **Download the installer from Ubuntu (https://www.ubuntu.com/download/server)**
2. **Write the installer to a USB stick (https://tutorials.ubuntu.com/tutorial/tutorial-create-a-usb-stick-on-macos#0)**
3. Plug the USB stick into the computer and boot to the installer (I have had a weird experience with NUC hardware where only some USB ports can boot from a USB drive, in case you find your machine doesn't boot up at first)

The install process starts from a menu where you can choose "Install Ubuntu Server" and presents you with a sequence of system configurations. I mostly stick with whatever the defaults are, but make sure that you configure the server to automatically install security upgrades. When you get to the "Software Selection" stage you should probably enable "OpenSSH Server" (press the space bar to toggle it), which will make it easier to manage the server remotely.

One key moment in the install process comes when the software is about to partition the disk and erase the contents: “Write the changes to disks and configure LVM?” You will have to select “yes” (which is not the default) and be sure you have the SSD selected, not the USB thumb drive. It will ask you a second time “Write the changes to disks?” Assuming the disk doesn’t have anything on it already, you can make these choices with confidence, with the understanding that you can always start over if something goes wrong.

You will be asked if you want to install the Grub boot loader (choose yes) and then which disk to install it onto (the SSD device). Finally the installer will ask you to remove the USB drive and then it will reboot your newly installed system.

Basic Ubuntu setup

Once the install has completed, you should get a login prompt at the console. Login with the account you set up before and then use the console session to figure out what the IP address is for the server. That will let you SSH in, which I find to be an easier way to work with the server, compared to directly typing into the keyboard console.

Use the `ifconfig` command to get the current network configuration. There should be a line like `inet addr:192.168.0.47` and that shows your current IP address. Yours will be different, depending on your particular network setup, but we’ll just assume in this guide that the IP is `192.168.0.47` as a placeholder.

The first thing we’ll do is upgrade to the latest versions of everything.

```
sudo apt-get update
sudo apt-get upgrade -y
```

Next we’ll add some basic utilities and secure the server with a `ufw` firewall. We’ll keep things simple here, but for a more comprehensive security baseline setup you may want to read **Bryan Kennedy’s “first five minutes” guide (https://plusbryan.com/my-first-5-minutes-on-a-server-or-essential-security-for-linux-servers)**.

```
sudo apt-get install -y git ufw fail2ban unattended-upgrades unzip make jq
sudo ufw allow 22
sudo ufw allow 80
sudo ufw allow 443
sudo ufw enable
```

Your firewall is now configured to listen on ports for SSH, HTTP, HTTPS, and will block network requests for anything else.

Where files go

We use the following convention for Who's On First:

- Code goes in `/usr/local/mapzen`
- Data goes in `/usr/local/data`

Here's how to prepare those base directories:

```
sudo mkdir -p /usr/local/mapzen
sudo mkdir -p /usr/local/data
sudo chown $USER /usr/local/mapzen
sudo chown $USER /usr/local/data
```

You can choose to install stuff elsewhere if you prefer, but the default configurations will expect things to exist in those base directories.

Download some data

Now we are getting to the good part: downloading WOF data! The Who's On First data repos are split up **by placetype (https://github.com/whosonfirst-data/)**. The data repository we'll be working with is `whosonfirst-data`, which only includes administrative placetypes. That includes records for cities, countries, neighbourhoods, etc. We've split other placetypes like venues, constituencies and postalcodes into their own separate repositories, but everything contained in this guide should work the same with them.

Each Who's On First repository contains two top level folders, `data` and `meta`. The `meta` folder has CSV summaries of all the GeoJSON files contained in the `data` directory. Those CSV meta files tend to get large, so we are currently storing them using **git lfs (https://git-lfs.github.com/)**. It's a service provided by GitHub that accommodates files over 100MB that get stored in the repo as stub files. Once we have git lfs installed, it will take care of pulling down those big CSV files from GitHub as a separate stage at the end of a `git pull`.

So before we can clone the data repositories, we'll need to **set up git lfs (https://github.com/git-lfs/git-lfs/wiki/Installation)**.

We are also ***in the process of phasing out git lfs-stored meta files (https://github.com/whosonfirst-data/whosonfirst-data/blob/master/meta/README.md#meta-files)****, in favour of having them generated on demand, but for now the* `git lfs` *dance is still an unfortunate dependency.*

```
curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | sudo
sudo apt-get install git-lfs
git lfs install
```

After you install git lfs, you may notice that your home directory has a new configuration in `~/.gitconfig` :

```
[filter "lfs"]
    process = git-lfs filter-process
    required = true
    clean = git-lfs clean -- %f
    smudge = git-lfs smudge -- %f
```

These filters are how the LFS system plugs into `git` commands.

This next part is going to take some time, so you may want to use a `screen` session (`tmux` is another popular alternative). I'm including a brief introduction to how `screen` works here.

```
screen -S download
```

This creates a new session called "download" that will keep your commands running in the background. It's especially helpful for when you can't stay logged in via SSH until a long task completes.

To "detach" from your session and keep it running behind the scenes, type `ctrl-A` followed by `D` (note to Mac users: this is the actual `ctrl` key, not `cmd`).

Then, to check up on its progress later, you can reattach by typing `screen -x download` . You can also use `screen -ls` to see what sessions you already have running.

You may want to read this **`screen` keyboard command cheat sheet (https://gist.github.com/fredrick/1216878)**, courtesy of **Fredrick Galoso (https://gist.github.com/fredrick)**.

Okay, now we can download some data!

```
cd /usr/local/data
git clone https://github.com/whosonfirst-data/whosonfirst-data.git
```

This should download about 37GB worth of Who's On First records, a little over 700,000 of worldwide administrative GeoJSON records.

Setting up Python

We are going to need some Python tools. It's important to mention that our Python libraries are a reference implementation and poorly documented. Recently we have been porting much of the functionality to Go, and we also intend to support Python 3 (and finally document things), but for now Python 2 is what works. Python has a lot of twisty dependency issues, so it can be weird and frustrating to set up, but with some patience we will get it all working.

```
sudo apt-get install -y python-pip python-pyparsing python-setuptools libpython-dev libss
sudo python -m pip install --upgrade --force pip
sudo pip install --upgrade setuptools
sudo pip install --upgrade pyopenssl ndg-httpsclient pyasn1 'requests[security]'
```

Now we can install the **WOF Python libraries (https://github.com/whosonfirst/py-mapzen-whosonfirst)**.

```
cd /usr/local/mapzen
git clone https://github.com/whosonfirst/py-mapzen-whosonfirst.git
cd py-mapzen-whosonfirst
sudo pip install --upgrade -r requirements.txt --process-dependency-links .
```

Now that we have WOF python tools we can start working with the records. Here is an example of a very basic crawl script you could use to iterate over every WOF record in a directory tree:

```
#!/usr/bin/env python

import mapzen.whosonfirst.utils

def cb(feature):
    print feature.keys()

source = "/usr/local/data/whosonfirst-data/data"
crawl = mapzen.whosonfirst.utils.crawl_with_callback(source, cb, inflate=True, multiproce
```

Note that you won't see any records in `/usr/local/data/whosonfirst-data` until the `git clone` command finishes in your `screen` session.

Setting up Elasticsearch

As we said at the top of this guide, we don't intend to choose your database, but **Elasticsearch (https://www.elastic.co/products/elasticsearch)** is what we use to run all of the tools in this guide.

We first need to install Java to run Elasticsearch. The first step will require an extra `[enter]` to confirm you're okay with adding an **apt Personal Package Archives source (https://askubuntu.com/questions/4983/what-are-ppas-and-how-do-i-use-them)** for Java.

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install -y oracle-java8-installer
```

The third step will prompt you to accept the Java legal terms.

Now we can install Elasticsearch itself.

```
wget https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/deb
sudo dpkg -i elasticsearch-2.4.6.deb
sudo systemctl enable elasticsearch.service
```

Next we will edit some configurations:

```
sudo nano /etc/elasticsearch/elasticsearch.yml
```

Add the following to the config:

```
cluster.name: whosonfirst
node.name: "Who's On First"
index.number_of_shards: 1
index.number_of_replicas: 0
```

Now we can start the Elasticsearch service:

```
sudo systemctl start elasticsearch
```

Give it 30 seconds or so to finish starting up, and then you should be able to `curl http://localhost:9200/` and get back a response ("you know, for search").

There's a more detailed discussion of setting up Elasticsearch in Ubuntu 16.04 in the excellent **DigitalOcean documentation (https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-elasticsearch-on-ubuntu-16-04)**.

Elasticsearch schemas

Elasticsearch is great at a lot of things, but it is really fussy about how you maintain its index schemas. By default it uses "magic" to infer datatypes of JSON properties when you index them, but when it guesses wrong the result can be very frustrating. As a result we need to be careful with managing our index schemas.

We keep the WOF Elasticsearch schemas in **a dedicated repo (https://github.com/whosonfirst/es-whosonfirst-schema)**.

```
cd /usr/local/mapzen
git clone https://github.com/whosonfirst/es-whosonfirst-schema.git
```

We have some synonyms that help out decipher incoming search queries, we need to copy those to the Elasticsearch config folder.

```
sudo mkdir -p /etc/elasticsearch/synonyms
cd /usr/local/mapzen/es-whosonfirst-schema/synonyms
sudo cp *.txt /etc/elasticsearch/synonyms/
```

Now we can set up the schema for the Spelunker.

```
cd /usr/local/mapzen/es-whosonfirst-schema
cat schema/2.4/mappings.spelunker.json | curl -X PUT http://localhost:9200/spelunker_2017
curl -X POST http://localhost:9200/_aliases -d '{ "actions": [ { "add": { "alias": "spelu
```

We have actually done two things:

1. Created an index called `spelunker_20171221`
2. Aliased it to respond to `spelunker`

Using an alias makes upgrading the schema easier in the future.

Let's check to make sure the index and alias are set up properly:

```
curl -s http://localhost:9200/_aliases | jq
```

You should see a response like:

```
{
  "spelunker_20171221": {
    "aliases": {
      "spelunker": {}
    }
  }
}
```

Index the data

The next logical step is to index the data into Elasticsearch, but there is a chance you may still be waiting on the `git clone` command to finish. This might be a good time to check on the download we had running in a `screen` session from earlier: `screen -x download` . Remember, to detach from the session press `ctrl-A` then `D` .

If it hasn't finished, continue on with the next steps and circle back to this later.

Once `git` has finished cloning, we can index the records into Elasticsearch. This will also take a while to complete, so you may want to just keep the `screen` session going.

```
cd /usr/local/data/whosonfirst-data
wof-es-index -s . --index=spelunker -b
```

Once this completes, you'll be able to make requests to Elasticsearch to inspect individual records like this:

```
curl -s http://localhost:9200/spelunker/locality/102027181 | jq
```

That should give you the GeoJSON for **Shanghai (https://whosonfirst.mapzen.com/spelunker/id/102027181/)**.

Installing the Spelunker

Before we spin up the Spelunker itself, there are *more* Python dependencies we need to install first.

```
sudo apt-get update
sudo apt-get install -y gunicorn python-gevent gdal-bin
sudo pip install flask flask-cors pycountry
```

Now we can download and set up the Spelunker web app.

```
cd /usr/local/mapzen
git clone https://github.com/whosonfirst/whosonfirst-www-spelunker.git
cd whosonfirst-www-spelunker
./ubuntu/setup-spelunker.sh
./ubuntu/setup-gunicorn.sh
```

The Spelunker itself is a simple **Flask application (http://flask.pocoo.org/)**. Flask applications implement the **Web Server Gateway Interface (https://www.python.org/dev/peps/pep-0333/)** which allow requests to be handled a more robust web server application, like **gunicorn (http://gunicorn.org/)**. Finally, for our purposes we route requests to the `gunicorn` web server through the `nginx` proxy server, discussed below.

Basically gunicorn is the long underwear and nginx is the down parka that your Flask application (the Spelunker) wears when it goes outside in to the cold winter of the Internet. An unfortunate side-effect of this otherwise fruitful relationship is that there a few too many settings and different config files that need to be managed in order for everything to work smoothly. Which is to say:

There are a handful of config files in the `config` folder you may want to look at, but we will leave them with their default settings.

There is one JavaScript config file we need to set up.

```
cd /usr/local/mapzen/whosonfirst-www-spelunker/www/static/javascript
cp mapzen.whosonfirst.config.js.example mapzen.whosonfirst.config.js
nano mapzen.whosonfirst.config.js
```

Look for `var api_key = 'mapzen-xxxxxx';` and replace the key with your own Mapzen API key. You can register one from the **Mapzen dashboard (https://mapzen.com/dashboard)**.

Next we will set up a log file for the Spelunker.

```
sudo touch /var/log/spelunker
sudo chown www-data:www-data /var/log/spelunker
```

Let’s try starting up the Spelunker in debug mode. (Note that we intend to update this `init.d` script to use something more modern.)

```
sudo /etc/init.d/whosonfirst-www-spelunker.sh debug
```

You will very likely get an error at this point: `Error: class uri 'egg:gunicorn#gevent' invalid or not found`

We are still figuring out why `gevent` isn't working on platforms like the NUC, so the interim solution is to just comment it out. If you happen to know *why* gevent isn't working for us, we'd be very interested to know why.

```
cd /usr/local/mapzen/whosonfirst-www-spelunker/config
nano whosonfirst-www-spelunker-gunicorn.cfg
```

Look for this line and add a `#` comment to disable it:

```
# worker_class = "egg:gunicorn#gevent"
```

Then try again running in debug mode.

```
sudo /etc/init.d/whosonfirst-www-spelunker.sh debug
```

If that works, we can start the service to run in the background.

So, yes, `init.d` scripts have been deprecated in Ubuntu 16.04. We are still finding our way to the right **Systemd equivalents (https://www.digitalocean.com/community/tutorials/systemd-essentials-working-with-services-units-and-the-journal)**, so for now we will kind of hold our nose and pretend this is just fine. The first `killall` command is for shutting down any previously-running instances of the Spelunker.

```
sudo killall gunicorn
sudo /etc/init.d/whosonfirst-www-spelunker.sh start
```

You should now be able to see the service running locally.

```
curl http://localhost:7777/
```

Setting up nginx

Now that we have the Spelunker working we need a web proxy to send requests for `/spelunker` to `localhost:7777`. We'll be using **nginx (https://www.nginx.com/resources/wiki/)** to handle this.

But while we are here configuring nginx, we'll set it up for the rest of the WOF stack too.

- `/` - served statically from `/usr/local/data/whosonfirst-www/www`
- `/data` - raw data served statically from `/usr/local/data/whosonfirst-data/data`
- `/spelunker` - proxied to `localhost:7777`
- `/api` - proxied to `localhost:8888`

- `/boundaryissues` - proxied to `localhost:9999`

And that first path doesn’t exist yet, so let’s clone the Who’s On First website repo.

```
cd /usr/local/mapzen
git clone https://github.com/whosonfirst/whosonfirst-www.git
```

This gives us a basic website with lots of Who’s On First documentation, available from the server’s homepage.

Okay, now let’s install nginx.

```
sudo apt-get update
sudo apt-get install -y nginx
cd /etc/nginx/sites-available
```

Now create a new config file `whosonfirst` .

```
sudo nano whosonfirst
```

Add the following configurations to it:

```
server {
    listen 80;
    server_name _;
    index index.html;

    root /usr/local/mapzen/whosonfirst-www/www;

    location / {
        try_files $uri $uri/ =404;
    }

    location /data {
        root /usr/local/data/whosonfirst-data/data;
    }

    location /spelunker {
        proxy_pass http://localhost:7777;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-Proxy-Path /spelunker; # this bit is important (tomorrow)
        proxy_set_header X-Script-Name /spelunker; # this bit is important (today)
        # https://github.com/whosonfirst/whosonfirst-www-spelunker/blob/master/www/server
    }

    location /api {
        proxy_pass http://localhost:8888;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-Proxy-Path /api;
    }

    location /boundaryissues {
        proxy_pass http://localhost:9999;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-Proxy-Path /boundaryissues;
    }
}
```

Next we will enable the `whosonfirst` configuration.

```
cd /etc/nginx/sites-enabled
sudo rm default
sudo ln -s ../sites-available/whosonfirst whosonfirst
```

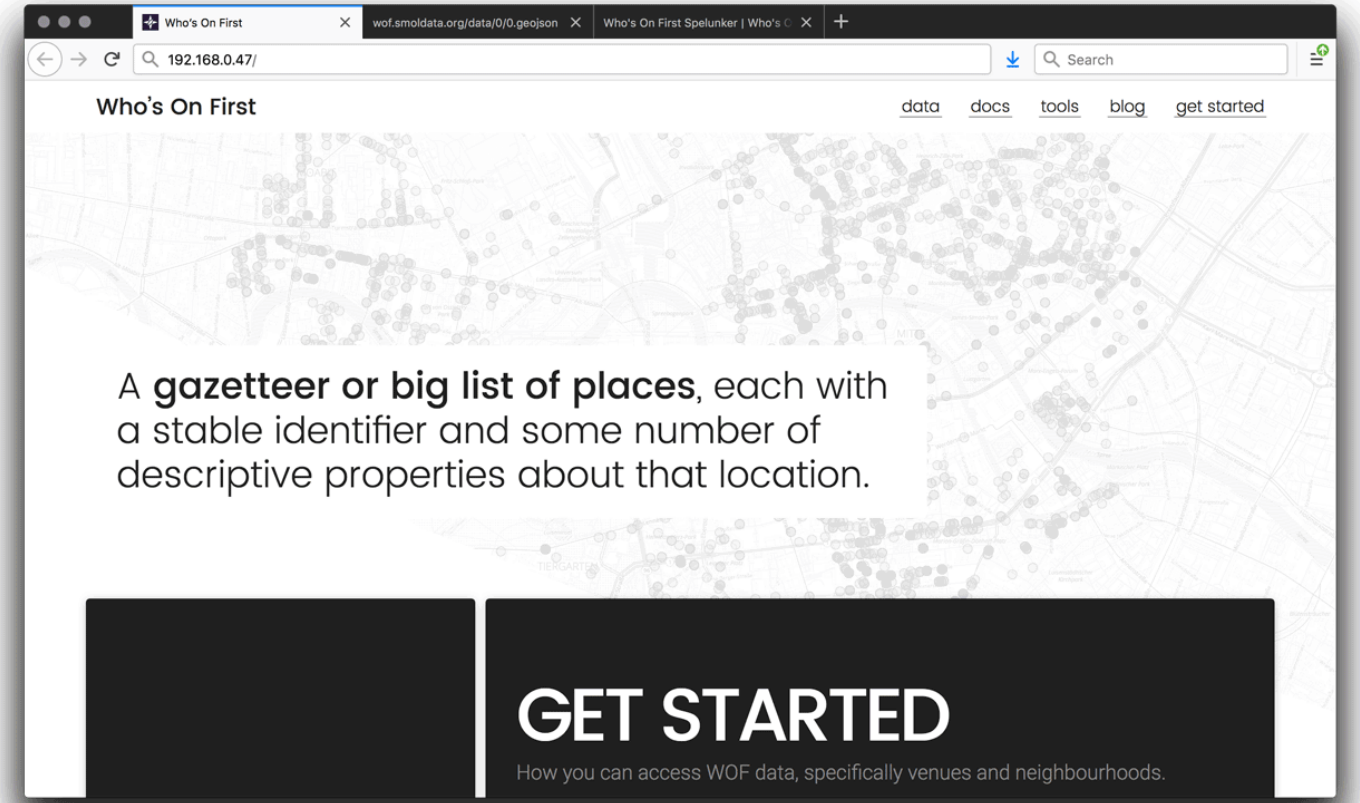
Now we can restart nginx for our new configurations to kick in.

```
sudo service nginx restart
```

Here’s a quick test that should indicate whether things are working:

```
curl -I http://localhost/
curl -I http://localhost/data/0/0.geojson
curl -I http://localhost/spelunker/
```

If each of those report back with `HTTP/1.1 200 OK` then you are in good shape. If the second request for `0.geojson` comes back with a 404 error, that may be because you’re still not done cloning data with `git`. If you want to test it out from a web browser, just go to **http://192.168.0.47/spelunker/ (http://192.168.0.47/spelunker/)** (substituting whatever your actual IP address is).


Who's On First
wof.smoldata.org/data/0/0.geojson
Who's On First Spelunker | Who's O
192.168.0.47/
Search
Who's On First
data
docs
tools
blog
get started

A **gazetteer or big list of places**, each with a stable identifier and some number of descriptive properties about that location.

GET STARTED

How you can access WOF data, specifically venues and neighbourhoods.

Stay tuned for part two

That concludes the first part of the WOF in a Box guide. We have some WOF data and a Spelunker tool to explore it with. The second installment will add a **REST API (https://github.com/whosonfirst/whosonfirst-www-api/)** and a **web-based CMS (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/)** to edit records with.

· 21 December 2017 ·

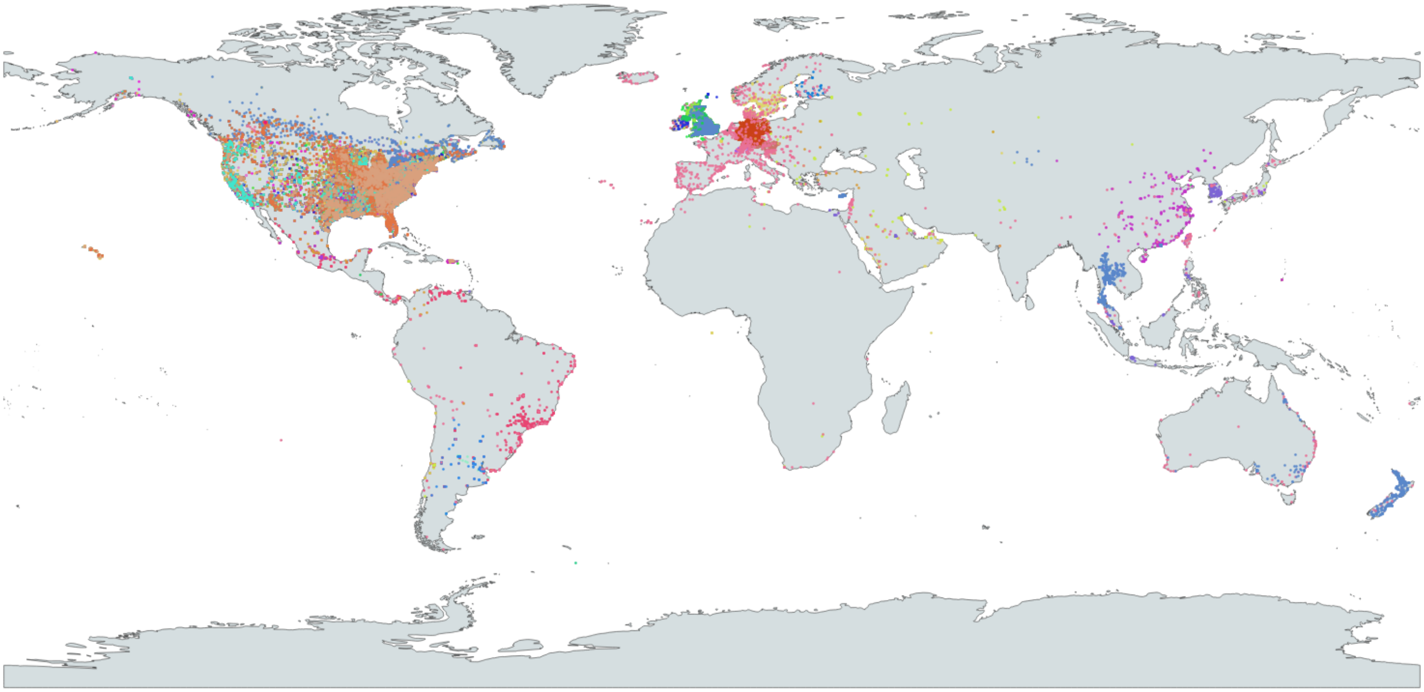
Dan Phiffer

Dan is a data carpenter, which tbh is kind of a made up title. But he likes it all the same.

© 2017 Mapzen

Scraping the web for All The Places

data (/tag/data)



Retail brands, chain stores, and other public and private venues have a vested interested in making sure people can find their brick and mortar locations. As a result, they frequently put the information on their website in the form of store locators or listings. Unfortunately, the data is almost never made available in a format that's easily machine-readable.

The new All The Places project extracts this useful location information into machine readable geographic data with **useful properties (https://github.com/alltheplaces/alltheplaces/blob/master/DATA_FORMAT.md)**. It consists of hundreds of website spiders that crawl the web looking for information to extract, translate into a better format, and the code to collate it all together into a data dump suitable for publication.

After all the spiders run, the result is posted to **alltheplaces.xyz (http://www.alltheplaces.xyz)** in GeoJSON format for public download. The last run from Dec. 21st, 2017 included **300,000+** lines (one for each store location) of newline delimited GeoJSON data. This data can then be used to build maps of your favorite stores, merged into gazeteers like **Who's On First (https://whosonfirst.mapzen.com/)**, and integrated into search engines like **Pelias (http://pelias.io/)**.

All The Places is an independent open source project similar to OpenAddresses in scope and resides **on GitHub (https://github.com/alltheplaces)**. It's easy to contribute to All The Places!

- Look through **the issues (https://github.com/alltheplaces/alltheplaces/issues)** to see if we've already tackled your favorite restaurant, salon, or clothing chain. If it's not there, simply add it as an issue.
- Are you intersted in web scraping or excited to learn how to write spiders? Pick one of **the open issues that are unassigned (https://github.com/alltheplaces/alltheplaces/issues?q=is%3Aopen+is%3Aissue+no%3Aassignee)** and **read the documentation (https://github.com/alltheplaces/alltheplaces/blob/master/README.md)** to get started writing your own spider.

· 22 December 2017 ·

Ian Dees

Ian is on the tiles team, building squares of map data. Ian ❤️maps and bleeds nodes, ways, and relations.

© 2017 Mapzen

Updating Who's On First Neighbourhoods - Part III

whosonfirst (/tag/whosonfirst) data (/tag/data)

WELCOME TO THE NEIGHBORHOOD
CRESTVIEW
BRENTWOOD

(https://mapzen-assets.s3.amazonaws.com/images/neighbourhood-updates-III/welcome_to_the_neighbourhood.jpg)

In two previous blog posts (**part one (https://mapzen.com/blog/sf-neighbourhood-updates/)**, **part two (https://mapzen.com/blog/neighbourhood-updates-two/)**), we discussed out efforts to update neighbourhood records in Who's On First in over 35 localities around the globe. Today, we're happy to report that we've now imported open neighbourhood data for over 83 localities, as well as all administrative records in the Netherlands!

We've continued to use our streamlined approach to importing **new sources (https://github.com/whosonfirst/whosonfirst-sources/tree/master/sources)** and adding **new properties (https://github.com/whosonfirst/whosonfirst-**

properties/tree/master/aliases) to existing records. We've made substantial gains since our last neighbourhood update eight months ago.

Now, we'd like to take a moment to outline our most recent efforts.

So, what's been updated?

In addition to the previous 35 localities that we've updated, we have imported new geometries and additional properties for microhoods, neighbourhoods, macrohoods, boroughs, localities, and some localadmin records in the following areas:

- All (!) of **the Netherlands (https://whosonfirst.mapzen.com/spelunker/download/85633337/)**
- **Toronto, Canada (https://whosonfirst.mapzen.com/spelunker/download/101735835/)**
- **Calgary, Canada (https://whosonfirst.mapzen.com/spelunker/download/890458845/)**
- **Vancouver, Canada (https://whosonfirst.mapzen.com/spelunker/download/101741075/)**
- **Burnaby, Canada (https://whosonfirst.mapzen.com/spelunker/download/101741055/)**
- North Vancouver, Canada (**City (https://whosonfirst.mapzen.com/spelunker/download/101741057/)** and **District (https://whosonfirst.mapzen.com/spelunker/download/101741071/)**)
- **New Westminster, Canada (https://whosonfirst.mapzen.com/spelunker/download/101741077/)**
- **Richmond, Canada (https://whosonfirst.mapzen.com/spelunker/download/101741081/)**
- **Surrey, Canada (https://whosonfirst.mapzen.com/spelunker/download/101741073/)**
- **UBC-Point Grey, Canada (https://whosonfirst.mapzen.com/spelunker/download/1108951499/)**
- **West Vancouver, Canada (https://whosonfirst.mapzen.com/spelunker/download/101741069/)**
- **Edmonton, Canada (https://whosonfirst.mapzen.com/spelunker/download/890458485/)**
- **Sherwood Park, Canada (https://whosonfirst.mapzen.com/spelunker/download/890456619/)**
- **St. Albert, Canada (https://whosonfirst.mapzen.com/spelunker/download/890458621/)**)
- **Victoria, Canada (https://whosonfirst.mapzen.com/spelunker/download/101740879/)**

- **Regina, Candada (https://whosonfirst.mapzen.com/spelunker/download/101739917/)**
- **Winnipeg, Canada (https://whosonfirst.mapzen.com/spelunker/download/101734959/)**
- **Saskatoon, Canada (https://whosonfirst.mapzen.com/spelunker/download/101740015/)**
- **Quebec City, Canada (https://whosonfirst.mapzen.com/spelunker/download/101737491/)**
- **Levis, Canada (https://whosonfirst.mapzen.com/spelunker/download/101737987/)**
- **Ottawa, Canada (https://whosonfirst.mapzen.com/spelunker/download/101735873/)**
- **Gatineau, Canada (https://whosonfirst.mapzen.com/spelunker/download/101737759/)**
- **Montreal, Canada (https://whosonfirst.mapzen.com/spelunker/download/101736545/)** (and smaller, surrounding localities)
- **Laval, Canada (https://whosonfirst.mapzen.com/spelunker/download/101737759/)**
- **Longueuil, Canada (https://whosonfirst.mapzen.com/spelunker/download/101738793/)**
- **Buenos Aires, Argentina (https://whosonfirst.mapzen.com/spelunker/download/421174285/)** (Completed by an external contributor, shout out to **@mfogel (https://www.github.com/mfogel)**!)
- **Madrid, Spain (https://whosonfirst.mapzen.com/spelunker/download/101748283/)**
- **Leganés, Spain (https://whosonfirst.mapzen.com/spelunker/download/101748281/)**
- **Mostoles, Spain (https://whosonfirst.mapzen.com/spelunker/download/101748289/)**
- **Fuenlabrada, Spain (https://whosonfirst.mapzen.com/spelunker/download/101748297/)**
- **Barcelona, Spain (https://whosonfirst.mapzen.com/spelunker/download/101748323/)**
- **London, UK (https://whosonfirst.mapzen.com/spelunker/download/101750367/)**
- **Santa Barbara, California (https://whosonfirst.mapzen.com/spelunker/download/85921779/)**

Impressive, but what do these changes actually look like?

Below, we've created an interactive map with neighbourhood name labels for our updated records in the Netherlands in **Tangram (https://mapzen.com/tangram/play/)**.

Notice that different placetype layers as you zoom in. You'll see:

- Locality records at a min zoom of `6`, max zoom of `7`.

- Borough records at a min zoom of `7`, max zoom of `8`.
- Macrohood records at a min zoom of `8`, max zoom of `9`.
- Neighbourhood records at a min zoom of `9`, max zoom of `10`.
- Microhood records at a min zoom of `10`, max zoom of `14`.

Dive in and enjoy!

This map requires WebGL support, but your browser does not support WebGL or it is currently disabled.
Learn more.

Click to view the Netherlands in full screen ➶ (https://tangrams.github.io/tangram-frame/?url=https://mapzen-assets.s3.amazonaws.com/images/neighbourhood-updates-III/netherlands_admin.yaml#6.75/52.278/5.553)

Property edit work

In addition to all of these new, clean geometries, Who’s On First has also completed a substantial amount of work around the following properties for neighbourhood records:

- **mz:min_max (https://github.com/whosonfirst/whosonfirst-properties/blob/master/properties/mz/min_zoom.json)**
- **mz:max_max (https://github.com/whosonfirst/whosonfirst-properties/blob/master/properties/mz/max_zoom.json)**
- **lbl:latitude (https://github.com/whosonfirst/whosonfirst-properties/blob/master/properties/lbl/latitude.json)**
- **lbl:longitude (https://github.com/whosonfirst/whosonfirst-properties/blob/master/properties/lbl/longitude.json)**

Curated values for these properties allow us to be specific when displaying placetype labels in our housestyles, like **Bubble Wrap (https://mapzen.com/products/maps/bubble-wrap/#lat=37.7611&lng=-122.4419&z=13.0250)**. We've imported new, curated zoom level and location values for thousands of Who's On First records and plan to import many more.

What's Next?

We are currently working on importing new geometries and properties for Paris, which should be imported into Who's On First shortly. Stay tuned for more and thanks for reading!

Photo Credit: ***Nathan Russell, Flickr (https://flic.kr/p/51Rkya)***

· 22 December 2017 ·

Stephen Epps

Stephen is Mapzen's gardener and fixer-upper of geographic data.

Zach Singleton

Zach is a GIS enthusiast who happens to have an unhealthy relationship with all things spicy.

© 2017 Mapzen

WOF in a Box (part 2)

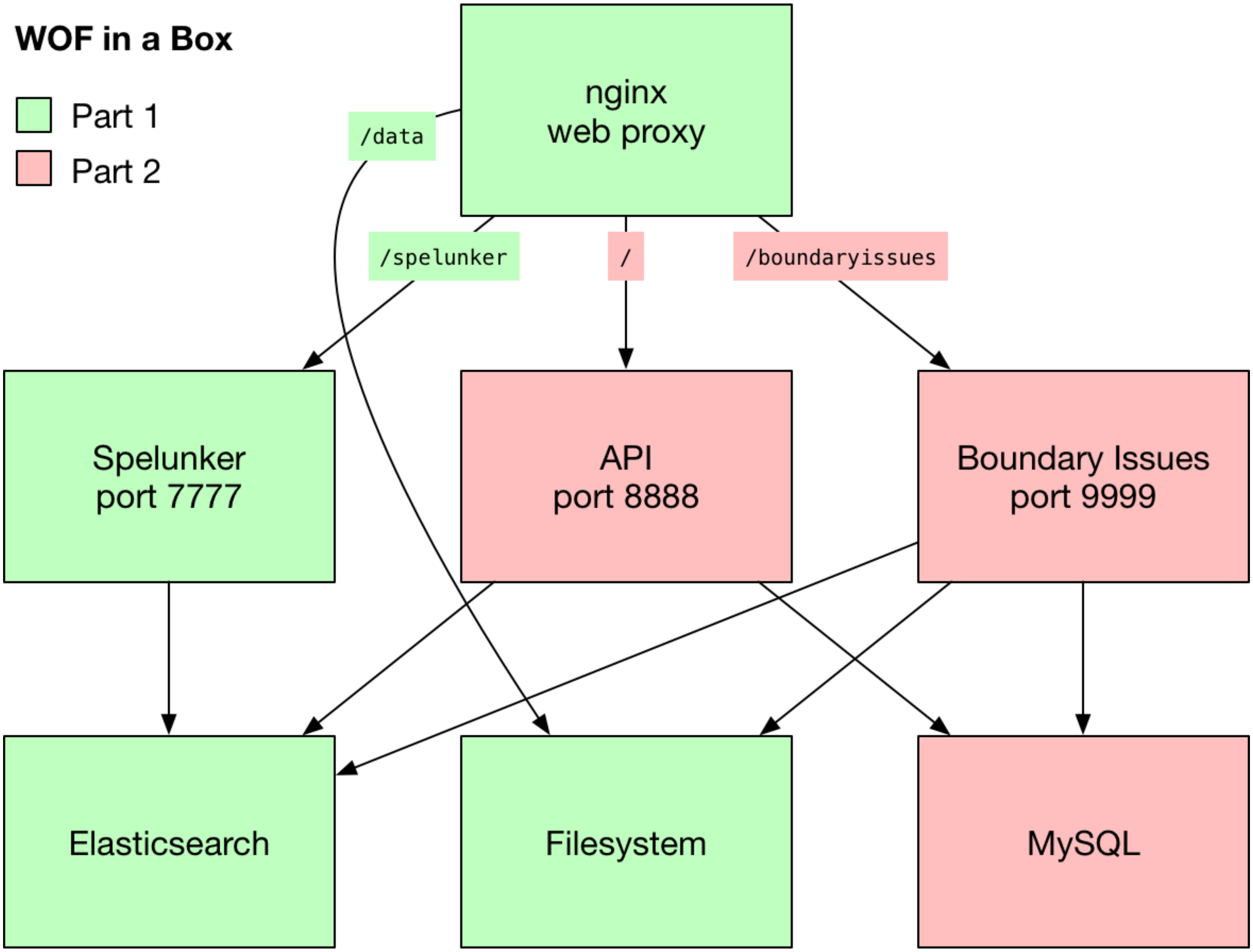
whosonfirst (/tag/whosonfirst)

This is the second of the two part guide to running ***Who's On First (https://whosonfirst.mapzen.com)*** *tools on your own hardware. You may want to read* ***part 1 (https://mapzen.com/blog/wof-in-a-box/)*** *first.*

Before we jump into installing the Who's On First **REST API (https://github.com/whosonfirst/whosonfirst-www-api)** and **editor (https://github.com/whosonfirst/whosonfirst-www-boundaryissues)**, let's briefly take stock of what we have running from the **first part (https://mapzen.com/blog/wof-in-a-box/)** of this guide.

- We have a git repo of **WOF administrative data (https://github.com/whosonfirst-data/whosonfirst-data)**
- Incoming web requests get proxied by **nginx (https://www.nginx.com/)**
- We can explore the data with our own installation of the **WOF Spelunker (https://github.com/whosonfirst/whosonfirst-www-spelunker/)**

Some of the stuff we set up in part one, like Elasticsearch and nginx, are dependencies for this second part of the stack too.


WOF in a Box
Part 1
Part 2
nginx
web proxy
/data
/spelunker
/
/boundaryissues
Spelunker
port 7777
API
port 8888
Boundary Issues
port 9999
Elasticsearch
Filesystem
MySQL

These next parts we're setting up run on a **LAMP stack (https://en.wikipedia.org/wiki/LAMP_(software_bundle))**, each built with **Flamework, a set of libraries and conventions (https://github.com/exflickr/flamework/blob/master/docs/philosophy.md)** originally created for the Flickr back-end. It still works just fine, although you can find some crufty bits if you poke around.

The two remaining web apps exist separately for now, but could very well be combined into a single codebase at some point in the future. The diagram shown above is rather simplified, and reflects a slight adjustment from the plan in the first part of this guide. Instead of API requests getting routed through `/api`, we will use the top-level `/` path.

The Who's On First documentation we used to have at the top-level `/` path is still **available online (https://whosonfirst.mapzen.com/)**, if not on your NUC.

Getting the time right

Before we start, there is one minor change from how we set up the server in **part one (https://mapzen.com/blog/wof-in-a-box/)** of the guide. During the Ubuntu server setup I mentioned that I just accept the defaults except for a handful important changes. One important change that I left out was to set the server's timezone to **UTC (Coordinated Universal Time) (https://en.wikipedia.org/wiki/Coordinated_Universal_Time)**.

We can easily revisit that step in the setup menu.

```
sudo dpkg-reconfigure tzdata
```

Then choose **None of the above**, then **UTC**. Then select **Ok**.

Let's err on the side of caution and install some software to ensure the server has the most up-to-date time set.

```
sudo apt-get update
sudo apt-get install -y ntpdate
```

Now we can update the time from the **National Institute of Standards and Technology (https://www.nist.gov/)**.

```
sudo ntpdate time.nist.gov
```

With that settled, let's continue.

(https://giphy.com/gifs/trolling-wtCFBQ0lNjxaU)

Setting up the API

Okay, let's begin setting up the API. I'm going to assume you're already SSH'd into the NUC machine.

```
cd /usr/local/mapzen
git clone https://github.com/whosonfirst/whosonfirst-www-api.git
```

Much of the API's setup process is automated, but if you look inside the **Makefile (https://github.com/whosonfirst/whosonfirst-www-api/blob/master/Makefile)** you can inspect each step in detail.

Since we are proxying requests with nginx, we will set things up without SSL support (for now).

```
cd /usr/local/mapzen/whosonfirst-www-api
make setup-nossl
```

You may be prompted for your password for `sudo` . You will also be asked to confirm that you want to install unattended upgrades (Yes) and which updates to install (the default is fine). Finally, you will choose a MySQL root password (and confirm it), then enter it once more to set up the database schema.

When all of this finishes your API setup will *almost* be in working order. Some things you should know about how the stack is set up:

- The API's Apache configuration is symlinked from `whosonfirst-www-api/config/whosonfirst-www-api-apache.conf`
- The database is called `wof_api` and is accessed by user `wof_api`
- All passwords and tokens are stored in `www/include/secrets.php`
- You will learn a lot by reading **`www/.htaccess` (https://github.com/whosonfirst/whosonfirst-www-api/blob/master/www/.htaccess)**
- There's also a great number of configuration tweaks contained in **`www/include/config.php` (https://github.com/whosonfirst/whosonfirst-www-api/blob/master/www/include/config.php)** (the convention is to set overrides in a separate file, we'll get to that down below)

Configuring Apache

We still need to adjust a couple configurations in Apache, to make the API run on port 8888.

```
cd /usr/local/mapzen/whosonfirst-www-api
nano config/whosonfirst-www-api-apache.conf
```

Edit the first line to use port 8888:

```
<VirtualHost *:8888>
```

Next, edit Apache's `ports.conf`.

```
sudo nano /etc/apache2/ports.conf
```

We're instructing Apache to listen on 8888 and also 9999 (for Boundary Issues). We can get rid of the existing configurations for port 80 and 443. Those ports are handled by nginx.

```
Listen 0.0.0.0:8888    # API
Listen 0.0.0.0:9999    # Boundary Issues
```

Let's restart Apache and check to make sure we can reach port 8888.

```
sudo service apache2 restart
curl -I http://localhost:8888/
```

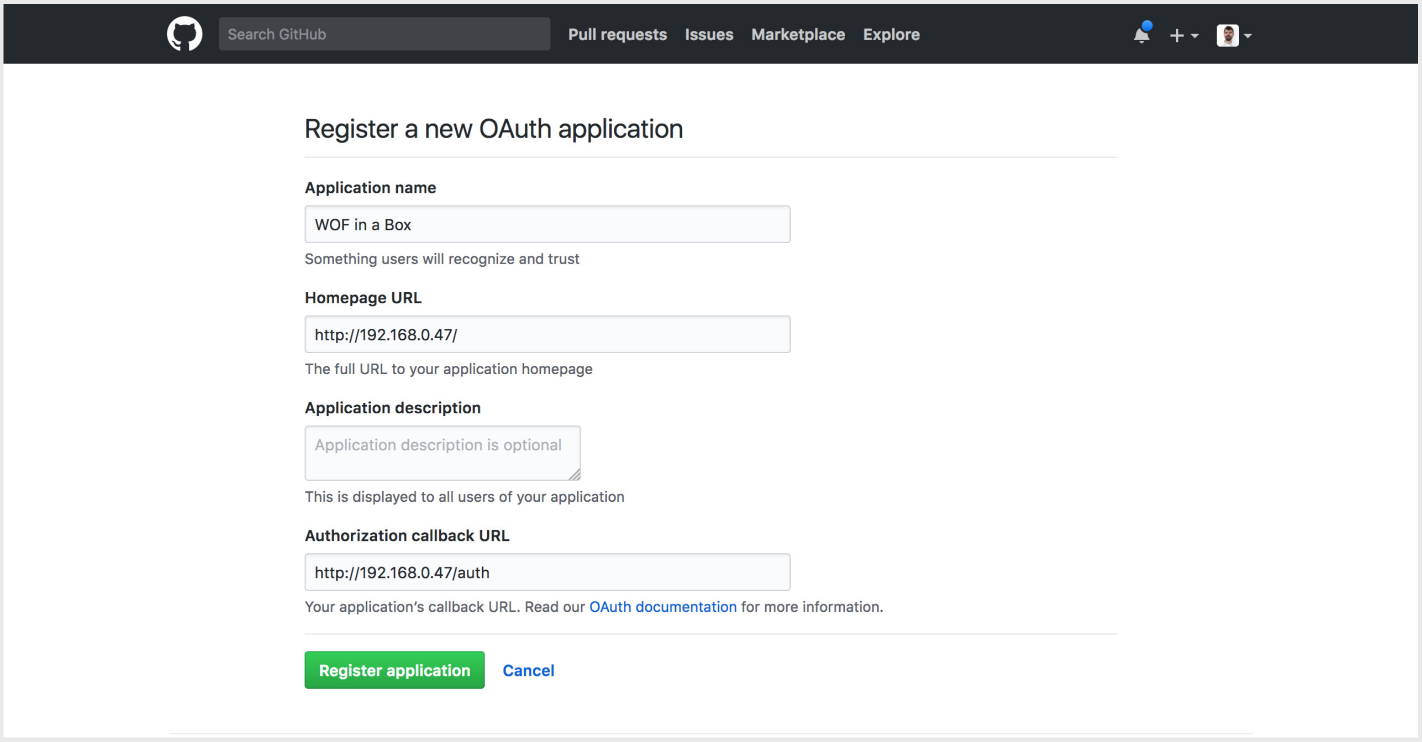
If you see `HTTP/1.1 200 OK` then you are in good shape.

GitHub OAuth application

We use GitHub's OAuth mechanism to authenticate users. We will need to set some things up at GitHub in order for that to work.

- Go to GitHub's **Developer settings (https://github.com/settings/developers)** and navigate to OAuth Apps
- Click the **New OAuth App (https://github.com/settings/applications/new)** button

- Enter an Application Name (`WOF in a Box` is a good choice)
- For the Homepage URL, enter your server's IP address, for example `http://192.168.0.47/`
- The Authorization callback URL is similar to the Homepage URL, but with an `/auth` path suffix, e.g., `http://192.168.0.47/auth`
- Click the Register Application button



Register a new OAuth application

Application name: WOF in a Box
Something users will recognize and trust

Homepage URL: http://192.168.0.47/
The full URL to your application homepage

Application description: Application description is optional
This is displayed to all users of your application

Authorization callback URL: http://192.168.0.47/auth
Your application's callback URL. Read our OAuth documentation for more information.

Edit `/usr/local/mapzen/whosonfirst-www-api/www/include/secrets.php` and copy/paste the GitHub OAuth tokens using the following configurations. You'll want to add this just before `# the end` .

```
$GLOBALS['cfg']['github_oauth_key'] = '(your client ID)';
$GLOBALS['cfg']['github_oauth_secret'] = '(your client secret)';
```

Secret hashes

While we're looking at this file, check to see if the first three configurations are set. If they have values like `You must set cfg.crypto_password_secret` that means we need to set them still. These are used for things like cookie sessions, passwords, and **CSRF (https://en.wikipedia.org/wiki/Cross-site_request_forgery)**-deterrent crumbs.

Generate a new secret hash for each of the three configurations, using the following script:

```
cd /usr/local/mapzen/whosonfirst-www-api
php bin/generate_secret.php
php bin/generate_secret.php
php bin/generate_secret.php
```

Those hashes get copied into `secrets.php` like this (your hashes will be different):

```
$GLOBALS['cfg']['crypto_cookie_secret'] = 'lfMi9sHoC2gnFKmMBgG8wUWqvqWO80eq';
$GLOBALS['cfg']['crypto_password_secret'] = 'afEp7fVgVe7TeO07jhud1y9opqgmhsmy';
$GLOBALS['cfg']['crypto_crumb_secret'] = 'u73GlqrDxqBOriFK921H8a0OsT8YZExz';
```

Configuring nginx

Before we can test everything, we will need to make one last adjustment to the nginx config. Instead of routing traffic from `/api` to port 8888, we will send everything from the top level `/` path.

```
sudo nano /etc/nginx/sites-enabled/whosonfirst
```

First, delete this block:

```
location / {
    try_files $uri $uri/ =404;
}
```

Then modify the API block to use `/` instead of `/api`. Notice there are two places that need editing, the `location /` part and the `proxy_set_header X-Proxy-Path /` part.

```
location / {
    proxy_pass http://localhost:8888;
    proxy_set_header Host $http_host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Scheme $scheme;
    proxy_set_header X-Proxy-Path /;
}
```

Let’s restart nginx to make those configurations take effect.

```
sudo service nginx restart
```

Logging in with GitHub

There is one last configuration we need to make for all of this to run on plain vanilla HTTP. It’s not so much that we wouldn’t want to run on HTTPS, just that we would need a public hostname, or a self-signed certificate.

If file `config_local.php` exists, it gets loaded in to override the default values defined in `config.php` .

```
cd /usr/local/mapzen/whosonfirst-www-api
nano www/include/config_local.php
```

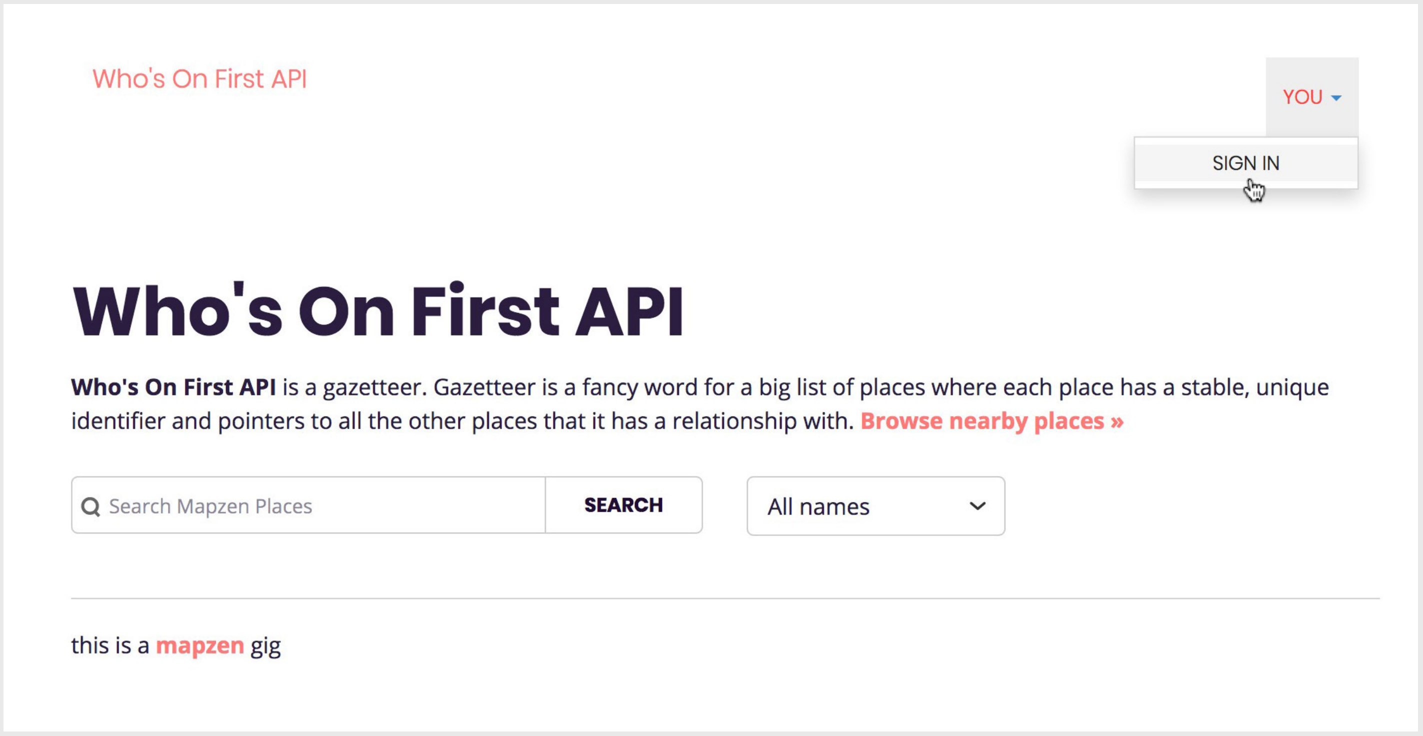
Add the following to disable HTTPS and, while we’re in here, use the right Elasticsearch index:

```
<?php

    $GLOBALS['cfg']['server_force_https'] = 0;
    $GLOBALS['cfg']['api_require_ssl'] = 0;
    $GLOBALS['cfg']['elasticsearch_spelunker_index'] = 'spelunker';
```

We don’t need a trailing `?>` since it is optional, and can lead to **unexpected side-effects (https://stackoverflow.com/questions/4410704/why-would-one-omit-the-close-tag)**.

Save your `config_local.php` configuration, and then load up your WOF in a Box's IP address (something like **http://192.168.0.47/ (http://192.168.0.47/)**) in a browser.

In the top-right corner, choose **Sign In** from the drop-down menu labeled **You**.

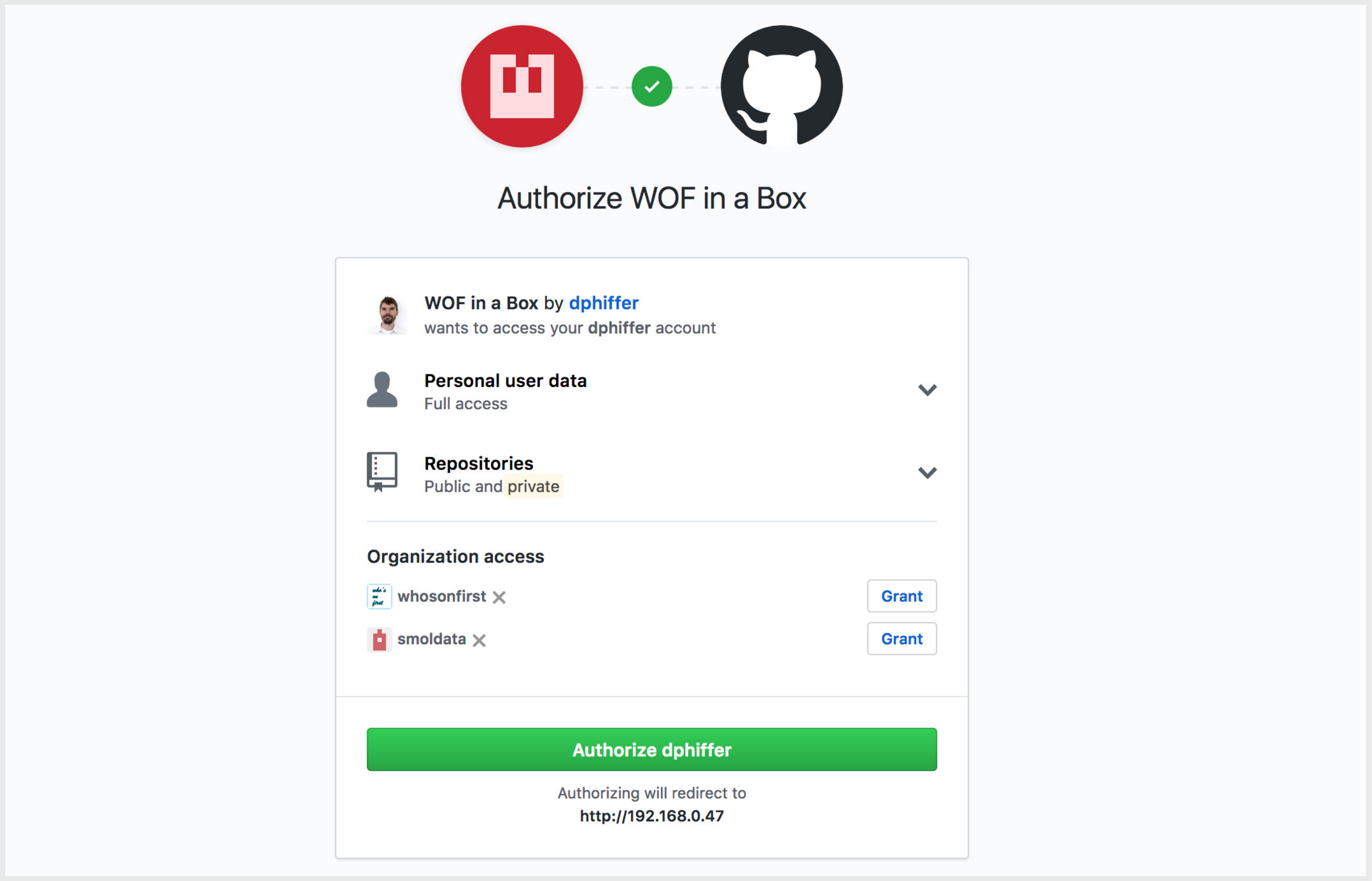


Who's On First API

Who's On First API is a gazetteer. Gazetteer is a fancy word for a big list of places where each place has a stable, unique identifier and pointers to all the other places that it has a relationship with. **Browse nearby places »**

this is a **mapzen** gig

GitHub will ask for permission to authenticate you with the OAuth application you just set up. Click the green button to login.



Authorize WOF in a Box

WOF in a Box by dphiffer
wants to access your **dphiffer** account

Personal user data
Full access

Repositories
Public and private

Organization access

Authorizing will redirect to
http://192.168.0.47

You should get redirected back to the WOF in a Box homepage. If all goes well you should see your name appear in the top-right drop-down menu. You have been OAuth'd by GitHub.

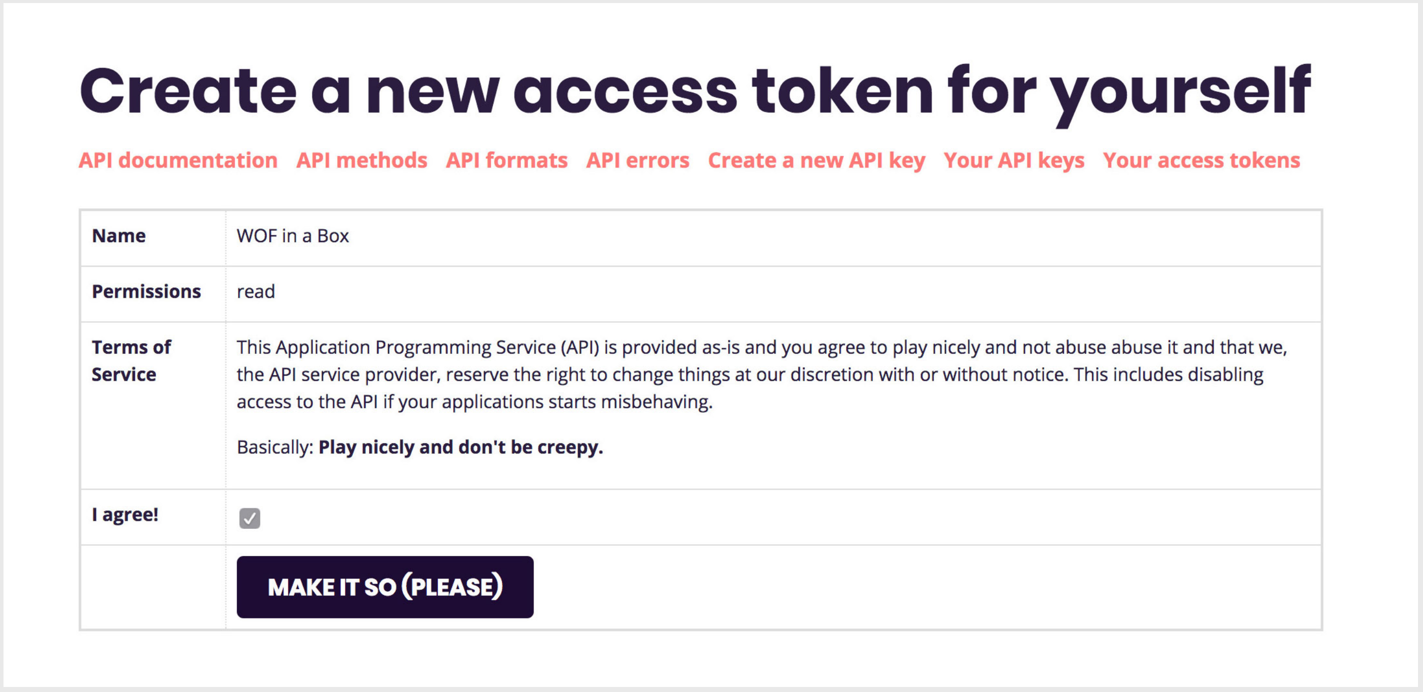
Create an access token

Go to the to-right drop-down menu and choose **Developer**. From here you can explore some of the API methods available, and configure OAuth access.

Click on **Your Access Tokens**, then **Create a new access token for yourself**.

Give your token a name ("WOF in a Box" is a safe choice), permissions ("Read" is good), and expiration ("until I revoke it").

Then you have to click one more **I agree!** checkbox and you should be set.

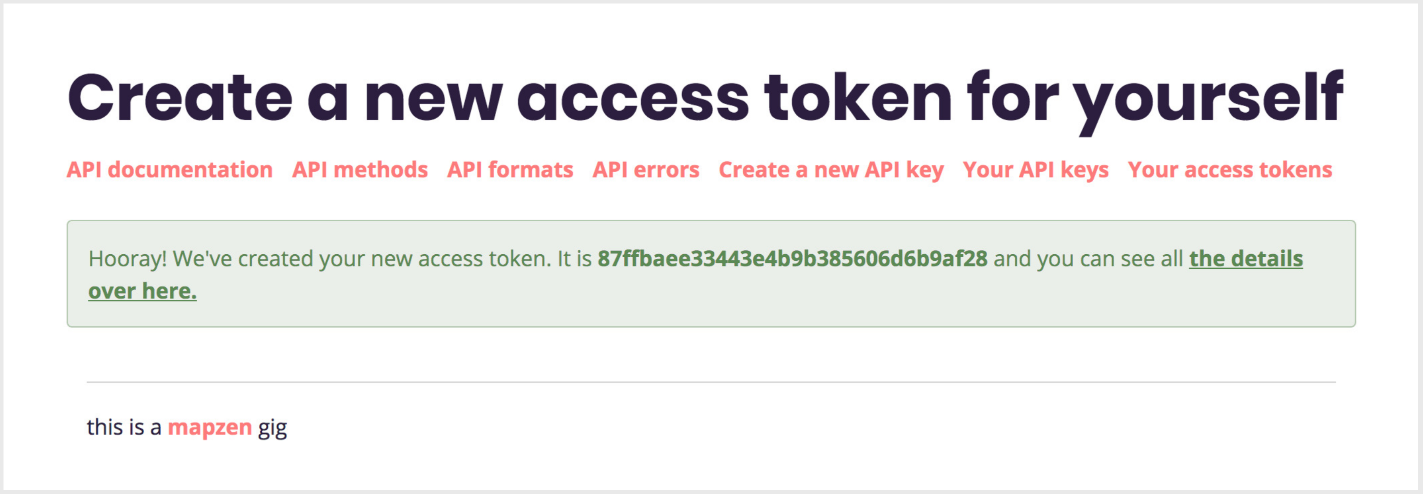


Create a new access token for yourself

API documentation API methods API formats API errors Create a new API key Your API keys Your access tokens

Name	WOF in a Box
Permissions	read
Terms of Service	This Application Programming Service (API) is provided as-is and you agree to play nicely and not abuse abuse it and that we, the API service provider, reserve the right to change things at our discretion with or without notice. This includes disabling access to the API if your applications starts misbehaving. Basically: **Play nicely and don't be creepy.**
I agree!	☑
	MAKE IT SO (PLEASE)

You should see a green “hooray” message, with a hexidecimal token for you to copy/paste.



Create a new access token for yourself

API documentation API methods API formats API errors Create a new API key Your API keys Your access tokens

Hooray! We've created your new access token. It is **87ffbaee33443e4b9b385606d6b9af28** and you can see all **the details over here.**

this is a mapzen gig

For the sake of having a realistic looking example token, we’ll go with `87ffbaee33443e4b9b385606d6b9af28` , but yours will be different.

Accessing the API

If you navigate to the **API methods** section of the Developer pages, you can try out some of the methods using the web-based API explorer.

Let’s try calling some API methods using `curl` .

```
curl -s "http://192.168.0.47/api/rest/?method=api.test.echo&access_token=87ffbaee33443e4b
```

You should see the following result:

```
{
  "method": "api.test.echo",
  "access_token": "87ffbaee33443e4b9b385606d6b9af28",
  "stat": "ok"
}
```

We can use a shell variable to store the IP address and access token, which should make the examples easier to copy/paste. Note, again, that your own IP and token will be different.

```
wof_api="http://192.168.0.47"
wof_token="87ffbaee33443e4b9b385606d6b9af28"
```

Now we should be able to test out the REST API. Let’s get a list of all the API method names.

```
curl -s "$wof_api/api/rest/?method=api.spec.methods&access_token=$wof_token" | jq "[ .met
```

Let’s load up some info about a specific WOF record.

```
curl -s "$wof_api/api/rest/?method=whosonfirst.places.getInfo&id=102030609&access_token=$
```

You should see a record for **Mumbai (https://whosonfirst.mapzen.com/spelunker/id/102030609/)**.

One last example, all the **microhoods (https://whosonfirst.mapzen.com/docs/placetypes/#microhood)** in **San Francisco (https://whosonfirst.mapzen.com/spelunker/id/85922583/)**:

```
curl -s "$wof_api/api/rest/?method=whosonfirst.places.getDescendants&id=85922583&placetyp
```

You should take some time to explore the API documentation, and try out the **standalone API Explorer desktop app (https://mapzen.com/blog/whosonfirst-api-explorer/)**. It has a settings pane that lets you change the API endpoint, so you can test out requests against your own WOF in a Box rig.

Important note: not all of the API methods will work for you at this point. We need some additional software for spatial queries and an additional Elasticsearch index for brands to work. Those can be subjects of some future WOF guide.

(https://giphy.com/gifs/cat-box-j1BQPAjNzKh9K)

Setting up Boundary Issues

We are in the home stretch. The final part of the stack is **our bespoke web-based WOF editor (https://mapzen.com/tag/boundaryissues/)**, called **Boundary Issues (https://github.com/whosonfirst/whosonfirst-www-boundaryissues/)**. It has a similar setup process as the API.

```
cd /usr/local/mapzen
git clone https://github.com/whosonfirst/whosonfirst-www-boundaryissues.git
```

Like the API, much of the setup has been automated with a Makefile to run a sequence of shell scripts. And as with the API, we will set things up without HTTPS, since we're proxying requests through nginx.

```
cd /usr/local/mapzen/whosonfirst-www-boundaryissues
make setup-nossl
```

You will be prompted for your `sudo` password, and then for the MySQL root password.

There is a second set of installation scripts that install everything required for offline tasks. The offline task functionality was designed to be an optional add-on to Boundary Issues, so it gets installed with a separate Makefile target.

```
make setup-offline
```

Most of the offline task dependencies are pulled in from modular single-purpose Flamework repos that take care of setting up each of:

- **redis (https://github.com/whosonfirst/flamework-redis)**
- **gearman (https://github.com/whosonfirst/flamework-gearman)**
- **logstash (https://github.com/whosonfirst/flamework-logstash)**

You may notice the process of setting up `gearman` has an exceptionally large set of dependencies. This is the unfortunate consequence of `apt-get install gearman` not working in Ubuntu 16.04. Instead we install a few low-level dependencies and then build it **from source (https://github.com/gearman/gearmand/)**. The process is automated by a shell script, so you shouldn't have to type anything extra, but you will notice a whole lot more scrolling by.

Also, for some reason `logstash` doesn't get validated during its installation, so you will need to manually confirm that you're okay with that.

Once all that finishes, things will be set up similarly to the API:

- The API's Apache configuration is symlinked from `whosonfirst-www-boundaryissues/config/whosonfirst-www-boundaryissues-apache.conf`
- The database is called `boundaryissues` and is accessed by user `boundaryissues`
- All passwords and tokens are stored in `www/include/secrets.php`

- Reading the bottom part of **`www/.htaccess`** **(https://github.com/whosonfirst/whosonfirst-www-boundaryissues/blob/master/www/.htaccess)** will show you all the URL path routing logic
- There's also an awful lot of configuration tweaks contained in **`www/include/config.php`** **(https://github.com/whosonfirst/whosonfirst-www-boundaryissues/blob/master/www/include/config.php)** (we'll set some overrides in a `config_local.php` down below)

Configuring Apache (again)

Like with the API, we need to adjust a couple configurations to make Boundary Issues run on port 9999.

```
cd /usr/local/mapzen/whosonfirst-www-boundaryissues
nano config/whosonfirst-www-boundaryissues-apache.conf
```

Edit the first line to use port 9999:

```
<VirtualHost *:9999>
```

Also, add this additional config after the `DocumentRoot` part, to get the `/boundaryissues` path to work properly:

```
Alias /boundaryissues /usr/local/mapzen/whosonfirst-www-boundaryissues/www
```

We already took care of `/etc/apache2/ports.conf` earlier, so we can just restart Apache and check to make sure we can reach port 9999.

```
sudo service apache2 restart
curl -I http://localhost:9999/
```

If you see `HTTP/1.1 200 OK` then take a moment to celebrate quietly before we move on.

Indexing Elasticsearch (again)

One of the quirks about the Who's On First ecosystem is that Boundary Issues and the Spelunker run on separate Elasticsearch indexes. This was intended to reduce the likelihood of accidentally screwing up live data in the Spelunker. We may combine them at some point, but for now it means we have some more schemas and indexing to take care of.

Let's start with the schemas, yes plural. Boundary Issues uses three of them! I'm assuming we already did all the other Elasticsearch setup from **part one (https://mapzen.com/blog/wof-in-a-box/)** of the guide.

```
cd /usr/local/mapzen/es-whosonfirst-schema
cat schema/2.4/mappings.boundaryissues.json | curl -X PUT http://localhost:9200/boundaryi
curl -X POST http://localhost:9200/_aliases -d '{ "actions": [ { "add": { "alias": "bound
cat schema/2.4/mappings.offline_tasks.json | curl -X PUT http://localhost:9200/offline_ta
curl -X POST http://localhost:9200/_aliases -d '{ "actions": [ { "add": { "alias": "offli
cat schema/2.4/mappings.audit_trail.json | curl -X PUT http://localhost:9200/audit_trail_
curl -X POST http://localhost:9200/_aliases -d '{ "actions": [ { "add": { "alias": "audit
```

That's a whole lot of schemas and aliases. Basically what we did was set up three new indexes, aliased to make future upgrades easier:

- `boundaryissues` aliased to `boundaryissues_20171229`
- `offline_tasks` aliased to `offline_tasks_20171229`
- `audit_trail` aliased to `audit_trail_20171229`

Now we can start the process of indexing the WOF records into the `boundaryissues` index. As with the Spelunker's Elasticsearch index, this will take some time to finish, so you may want to use `screen` for it.

```
cd /usr/local/data/whosonfirst-data
wof-es-index -s . --index=boundaryissues -b
```

Let that run in the background while we finish up the rest of the Boundary Issues setup.

Secret hashes and GitHub OAuth (again again)

Okay, so a lot of what's below may seem familiar after setting up the API. And I know you are eager to start editing WOF records. We are getting there!

(https://www.tumblr.com/search/cat%20in%20a%20box%20gif)

Unfortunately, because the API and Boundary Issues are separate apps, we need to set up a second GitHub OAuth application to handle redirects properly.

- Go to GitHub's **Developer settings (https://github.com/settings/developers)** and click the **New OAuth App (https://github.com/settings/applications/new)** button
- Enter another Application Name (`Boundary Issues in a Box` is a good choice)
- For the Homepage URL, enter your server's IP address, for example `http://192.168.0.47/boundaryissues`
- The Authorization callback URL is similar to the Homepage URL, but with an `/auth` path suffix, e.g., `http://192.168.0.47/boundaryissues/auth`
- Click the Register Application button

Now let’s open up `secrets.php` and insert those GitHub OAuth credentials.

```
$GLOBALS['cfg']['github_oauth_key'] = '(your client ID)';
$GLOBALS['cfg']['github_oauth_secret'] = '(your client secret)';
```

Next we’ll generate three new secret hashes, for the crypto hash configs:

```
cd /usr/local/mapzen/whosonfirst-www-boundaryissues
php bin/generate_secret.php
php bin/generate_secret.php
php bin/generate_secret.php
```

Those hashes get copied into `secrets.php` like this (your hashes will be different):

```
$GLOBALS['cfg']['crypto_cookie_secret'] = '9NBEy7d484AN3muSObRPsicoQIpjrhxQ';
$GLOBALS['cfg']['crypto_password_secret'] = 'HRoQz6jcuCB6lHhSmOhlWmfH3X7Nw9xa';
$GLOBALS['cfg']['crypto_crumb_secret'] = 'AxM2yCFsTwY9lVgyDUpJphrv4Qxf3gwI';
```

The pending folder

To accommodate the process of saving WOF records, we need to set up a www-writeable directory for incoming edits. It works kind of similarly to a Git index, with subfolders to keep the present edited state and a snapshot record of each edit.

The convention is to put the pending folder into the same structure as the other data repos.

```
sudo mkdir /usr/local/data/whosonfirst-pending
sudo chown www-data:www-data /usr/local/data/whosonfirst-pending
```

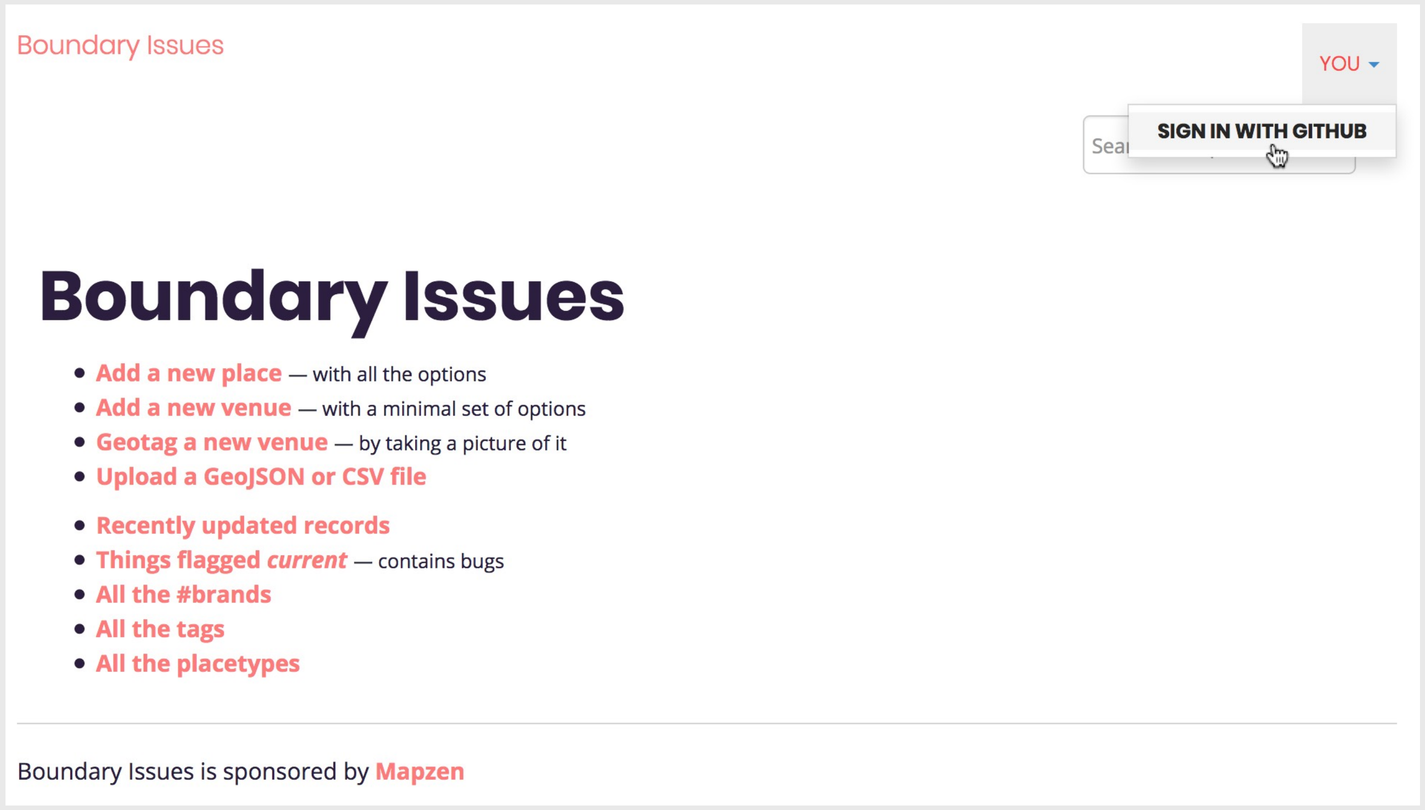
… and then symlink *that* into the Boundary Issues folder.

```
sudo ln -s /usr/local/data/whosonfirst-pending /usr/local/mapzen/whosonfirst-www-boundary
```

Let’s test things out

Now that we’ve set a whole bunch of Boundary Issues stuff… let’s maybe try it?

Go to `http://192.168.0.47/boundaryissues/` in a browser (again, with your IP subbed in), and see if it loads up. You should see something similar, but not quite the same as, the API’s homepage.



Boundary Issues

YOU

SIGN IN WITH GITHUB

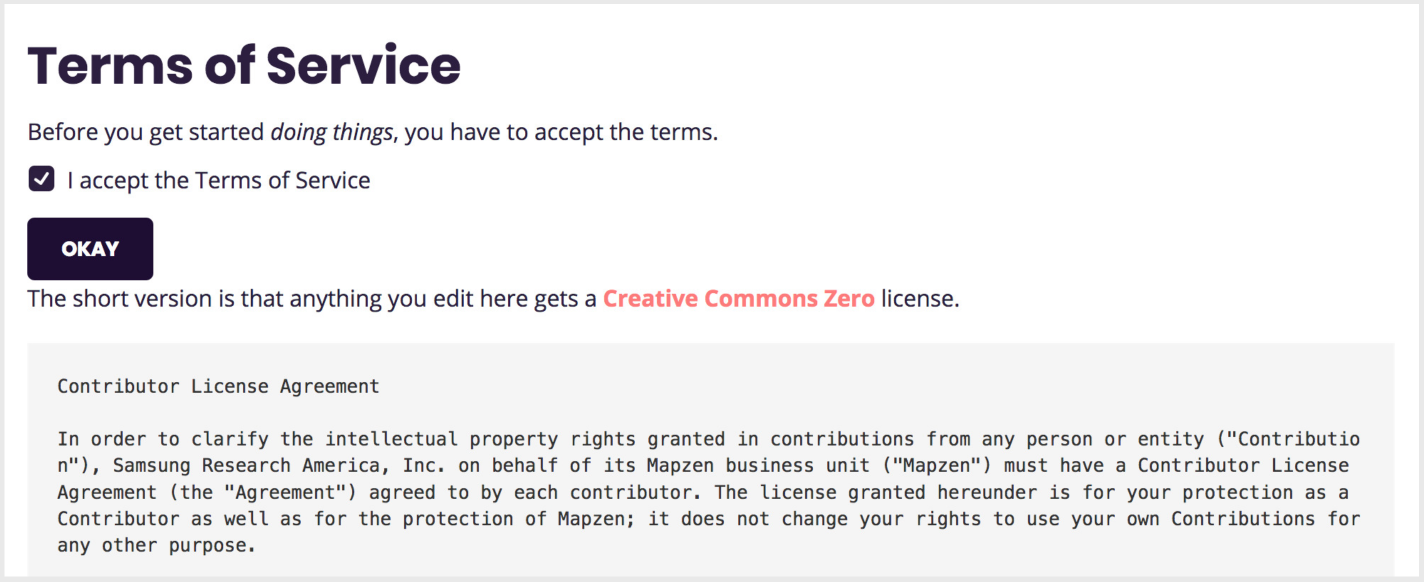
Boundary Issues

- Add a new place — with all the options
- Add a new venue — with a minimal set of options
- Geotag a new venue — by taking a picture of it
- Upload a GeoJSON or CSV file
- Recently updated records
- Things flagged *current* — contains bugs
- All the #brands
- All the tags
- All the placetypes

Boundary Issues is sponsored by Mapzen

Click on the same **You** menu and then **Sign in with GitHub**. You should see a similar OAuth confirmation page on GitHub, and then redirect back to the Boundary Issues homepage.

Once you get redirected back, you will have to accept some terms before you can continue.



Terms of Service

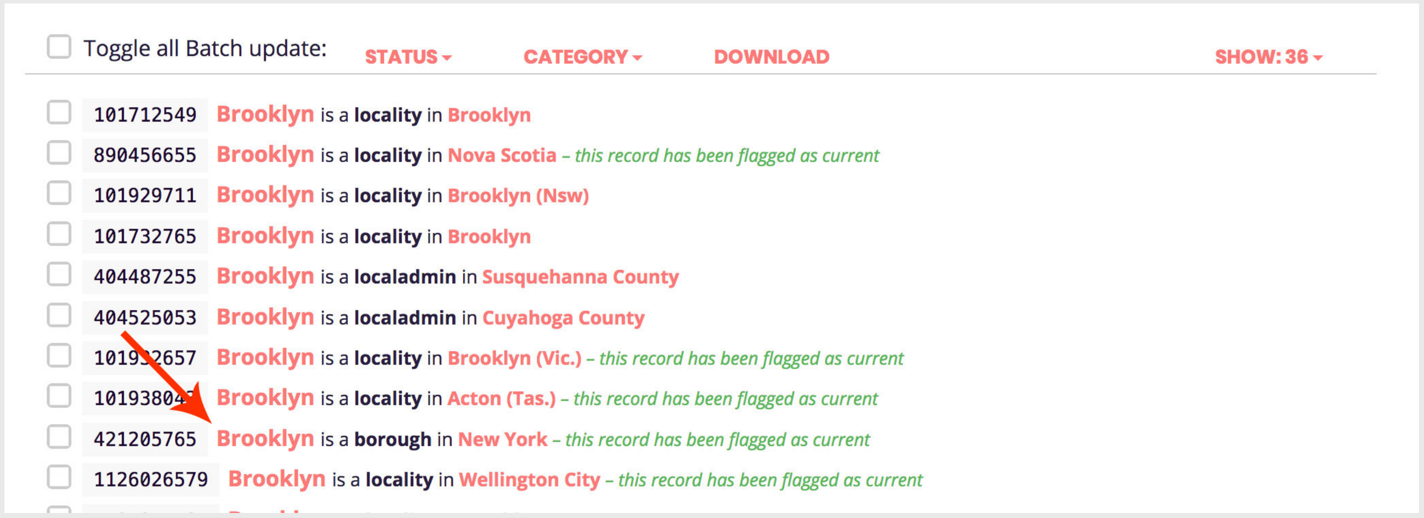
Before you get started *doing things*, you have to accept the terms.

☑ I accept the Terms of Service

The short version is that anything you edit here gets a **Creative Commons Zero** license.

```
Contributor License Agreement

In order to clarify the intellectual property rights granted in contributions from any person or entity ("Contributio
n"), Samsung Research America, Inc. on behalf of its Mapzen business unit ("Mapzen") must have a Contributor License
Agreement (the "Agreement") agreed to by each contributor. The license granted hereunder is for your protection as a
Contributor as well as for the protection of Mapzen; it does not change your rights to use your own Contributions for
any other purpose.
```

Let’s search for a record, for example **Brooklyn (https://whosonfirst.mapzen.com/spelunker/id/421205765/)**.



☐ Toggle all Batch update:

☐ 101712549 **Brooklyn** is a **locality** in **Brooklyn**

☐ 890456655 **Brooklyn** is a **locality** in **Nova Scotia** – *this record has been flagged as current*

☐ 101929711 **Brooklyn** is a **locality** in **Brooklyn (Nsw)**

☐ 101732765 **Brooklyn** is a **locality** in **Brooklyn**

☐ 404487255 **Brooklyn** is a **localadmin** in **Susquehanna County**

☐ 404525053 **Brooklyn** is a **localadmin** in **Cuyahoga County**

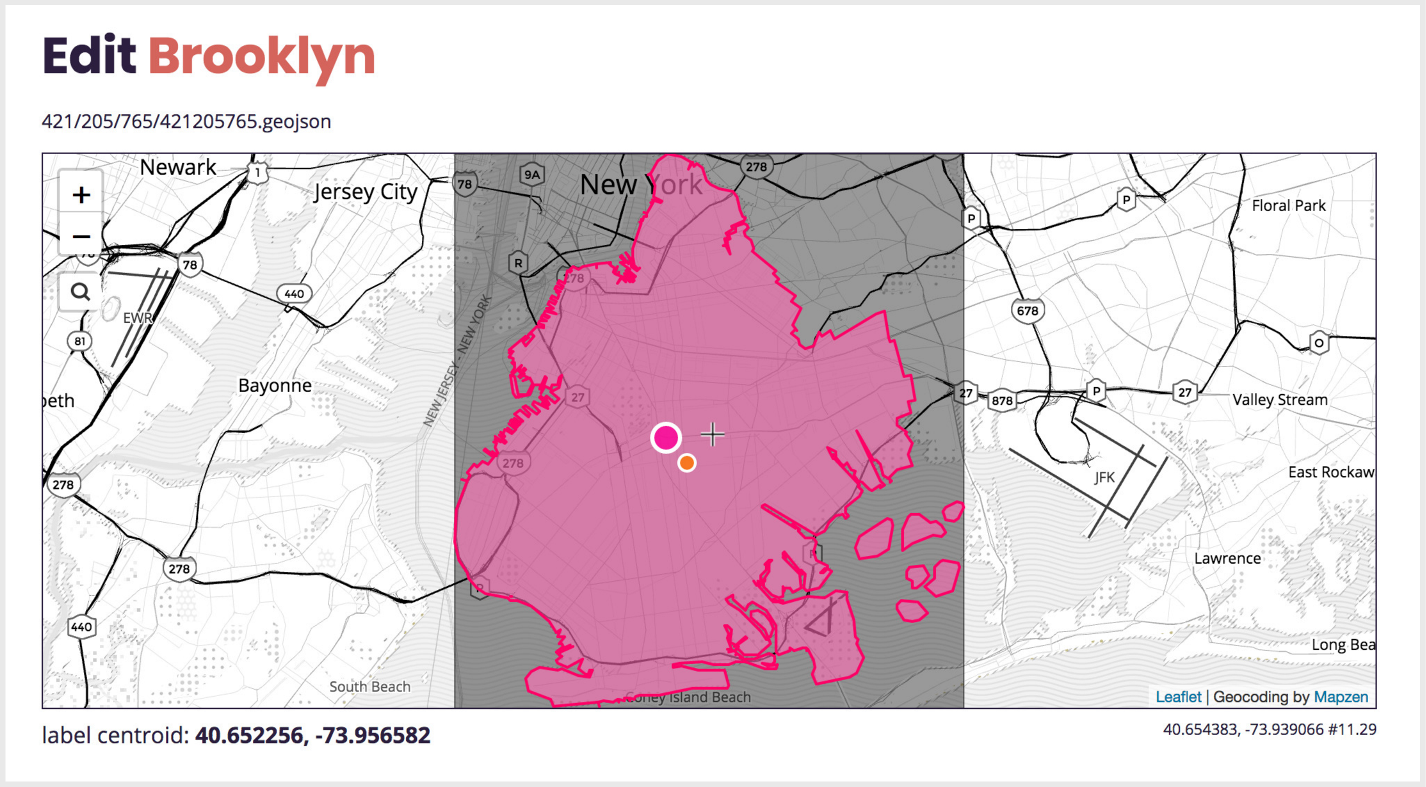
☐ 101932657 **Brooklyn** is a **locality** in **Brooklyn (Vic.)** – *this record has been flagged as current*

☐ 10193804[illegible] **Brooklyn** is a **locality** in **Acton (Tas.)** – *this record has been flagged as current*

☐ 421205765 **Brooklyn** is a **borough** in **New York** – *this record has been flagged as current*

☐ 1126026579 **Brooklyn** is a **locality** in **Wellington City** – *this record has been flagged as current*

You want to click on the one that’s a **Borough placetype (https://whosonfirst.mapzen.com/docs/placetypes/#borough)**.



Edit **Brooklyn**

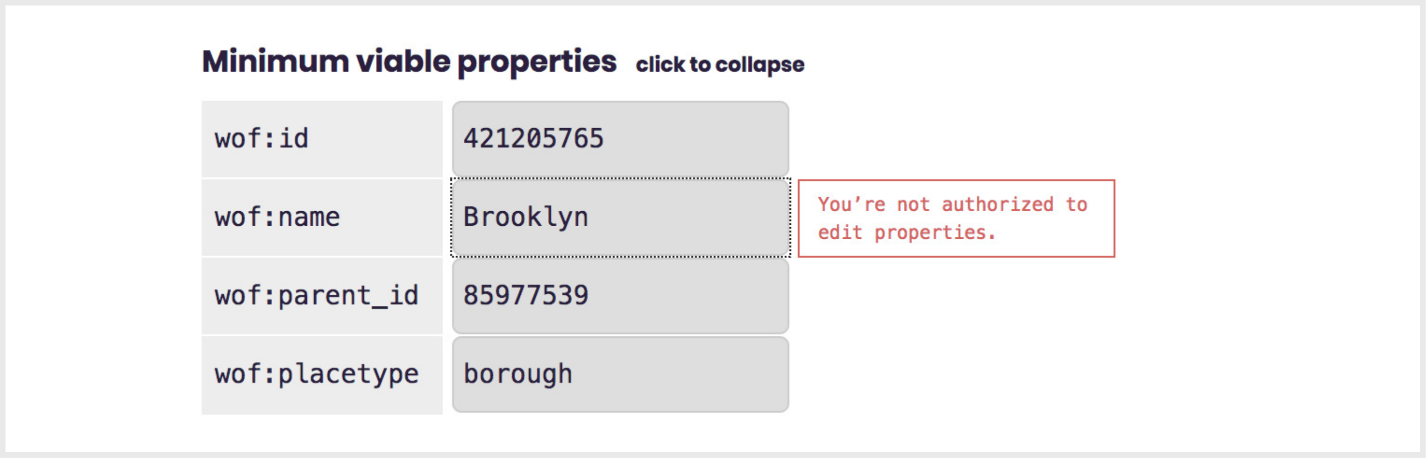
421/205/765/421205765.geojson

Newark
Jersey City
New York
Bayonne
NEW JERSEY - NEW YORK
EWR
South Beach
Coney Island Beach
Floral Park
Valley Stream
East Rockaw
JFK
Lawrence
Long Bea
Leaflet | Geocoding by Mapzen

label centroid: **40.652256, -73.956582**

40.654383, -73.939066 #11.29

Okay now we are getting somewhere, although…



Minimum viable properties **click to collapse**

`wof:id`	`421205765`
`wof:name`	`Brooklyn`
`wof:parent_id`	`85977539`
`wof:placetype`	`borough`

`You're not authorized to edit properties.`

… by default, new users don't have permissions to edit records.

Add a user role

One last setup task: we just need to add the role `admin` to your user account and you should be all set.

Here's how that works (again, swap in the email address you used to sign up for your GitHub account):

```
cd /usr/local/mapzen/whosonfirst-www-boundaryissues
php bin/add_user_role.php your.email@example.com admin
```

You should be prompted with a confirmation (press `Y` or just press enter).

There is actually *another configuration* we should set, inside a new `config_local.php` file.

```
cd /usr/local/mapzen/whosonfirst-www-boundaryissues/www/include
nano config_local.php
```

This will make Boundary Issues update the Spelunker's Elasticsearch index, upon saving.

```
<?php

    $GLOBALS['cfg']['enable_feature_index_spelunker'] = 1;
```

And *one more thing* (I know, this is complicated), we need to make sure we have the offline tasks running, since they take care of saving stuff after you hit the **Save** button.

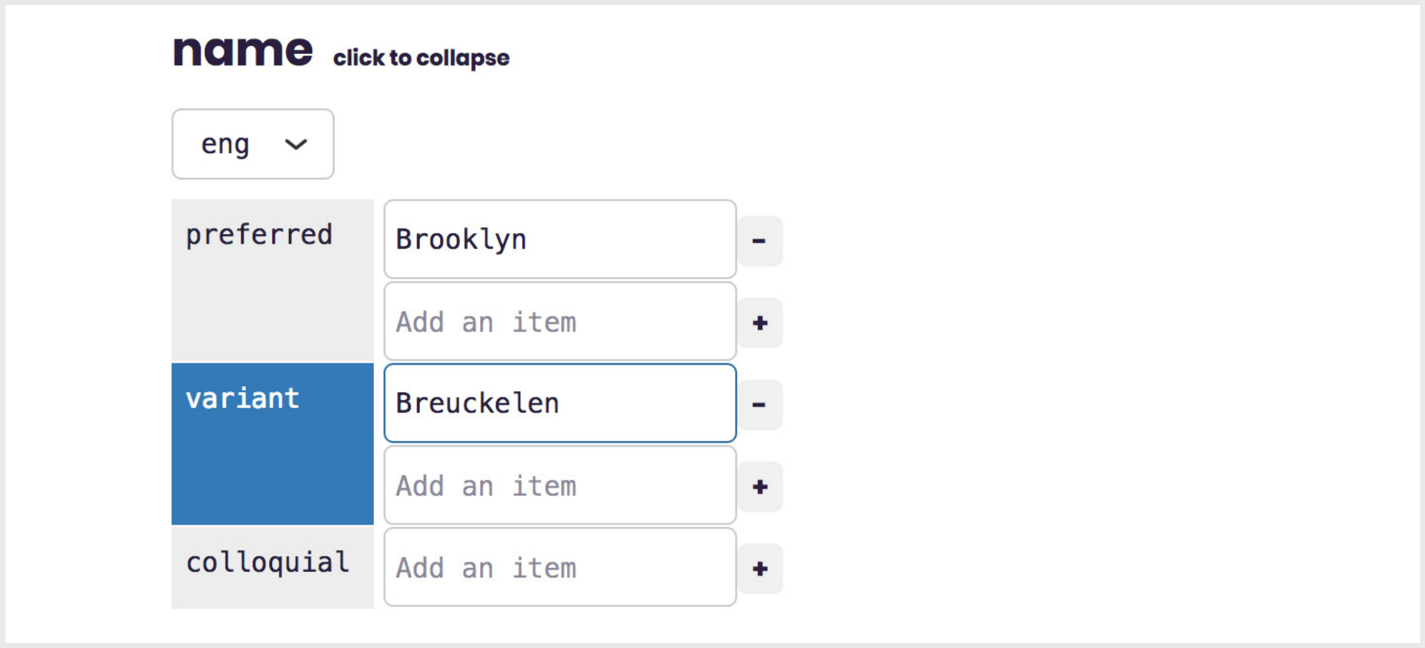
```
sudo /etc/init.d/gearmand start
sudo supervisorctl start all
```

Okay, now we should really be ready.

Edit a WOF record

Go back to the **Edit Brooklyn** page in Boundary Issues and reload it. You should be able to edit things now, yay!

Scroll down to the **name** section, and expand it open. Add the colonial name "Breuckelen" to the list of variant names (technically this should be under the Dutch language, not English, but one thing at a time).

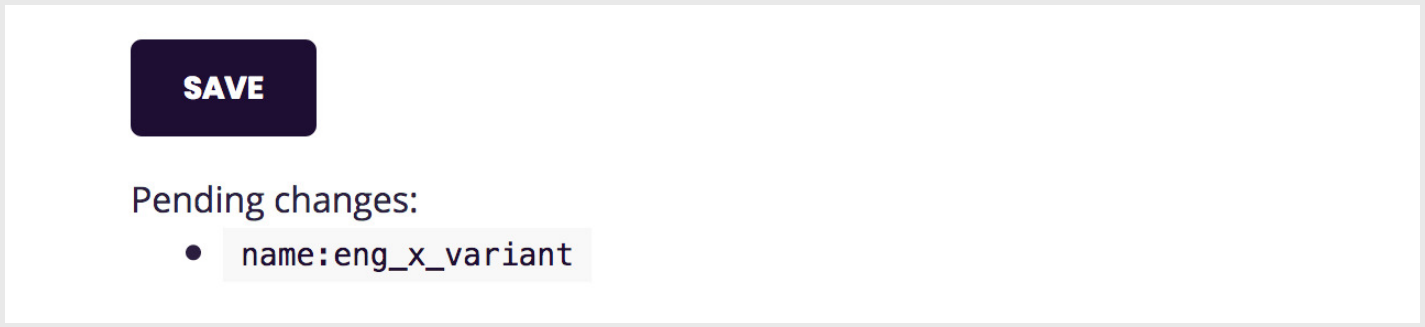


preferred: Brooklyn

variant: Breuckelen

colloquial:

Notice how the interface added a blue indicator next to the field? That helps you keep track of which things you’ve edited.

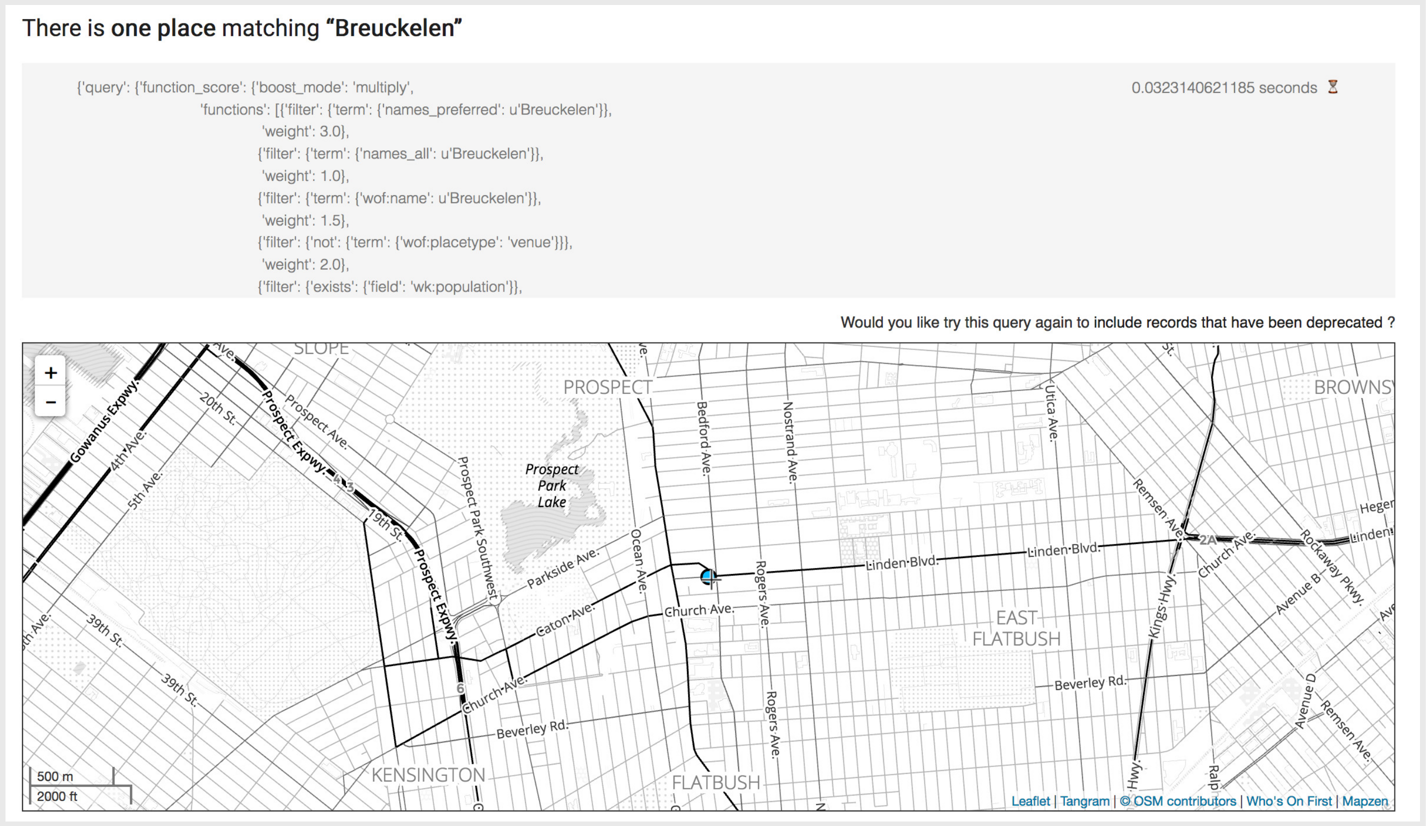


Pending changes:

- `name:eng_x_variant`

When you hit the save button, if all goes well, it should replace the list of changed properties with **Saved**. If it gets stuck on **Saving…** you may want to try again and keep an eye on the JavaScript console looking for error messages.

Finally, let’s try searching for updated name using the Spelunker.



There is **one place** matching **“Breuckelen”**

```
{'query': {'function_score': {'boost_mode': 'multiply',
                    'functions': [{'filter': {'term': {'names_preferred': u'Breuckelen'}},
                                   'weight': 3.0},
                                  {'filter': {'term': {'names_all': u'Breuckelen'}},
                                   'weight': 1.0},
                                  {'filter': {'term': {'wof:name': u'Breuckelen'}},
                                   'weight': 1.5},
                                  {'filter': {'not': {'term': {'wof:placetype': 'venue'}}},
                                   'weight': 2.0},
                                  {'filter': {'exists': {'field': 'wk:population'}},
```

0.0323140621185 seconds ⌛

Would you like try this query again to include records that have been deprecated ?

+
−
SLOPE
PROSPECT
BROWNS
Prospect Park Lake
Gowanus Expwy.
Prospect Expwy.
Prospect Ave.
20th St.
19th St.
4th Ave.
5th Ave.
39th St.
Prospect Park Southwest
Parkside Ave.
Ocean Ave.
Caton Ave.
Church Ave.
Beverley Rd.
Bedford Ave.
Nostrand Ave.
Rogers Ave.
Utica Ave.
Remsen Ave.
Linden Blvd.
Kings Hwy.
Rockaway Pkwy.
Avenue B
Avenue D
EAST FLATBUSH
KENSINGTON
FLATBUSH
500 m
2000 ft
Leaflet | Tangram | © OSM contributors | Who's On First | Mapzen

Hooray, you made an edit and it showed up in the Spelunker. There is more to all of this that I won’t go into in this blog post. Right now all the changes you make end up in `/usr/local/data/whosonfirst-pending`, without modifying the actual WOF Git repos. If we wanted to actually upstream the changes to GitHub, we would need to set up a new user on the server (we tend to go with `botsonfirst`) with credentials to push up to GitHub. And that user would execute the script `bin/save_pending.php` on a cron job.

There is also the whole subject of pipeline tasks vs. offline tasks, but maybe you can ask me about that over a beer some day.

MAKE GIFS AT GIFSOUP.COM

(http://i68.tinypic.com/2j3qvm1.gif)

Bundler config

Okay one final set of config changes before we call it a day. The Spelunker has a very handy tool called the **Bundler (https://mapzen.com/blog/bundler/)** and we should get it working with our new *WOF in a Box* stack.

SSH into the NUC server and edit the flask config.

```
cd /usr/local/mapzen/whosonfirst-www-spelunker/config/
nano whosonfirst-www-spelunker-flask.cfg
```

We'll make two edits here, the first enables the bundler feature:

```
enable_feature_bundler=1
```

The second will swap where the Spelunker looks for its WOF records from (once again, with your own IP swapped in):

```
data_root=https://192.168.0.47/data
```

There are a couple more edits to another config file: `mapzen.whosonfirst.config.js`

```
cd /usr/local/mapzen/whosonfirst-www-spelunker/www/static/javascript/
nano mapzen.whosonfirst.config.js
```

We are going to scroll down to the bottom and edit this part:

```
if (mapzen.whosonfirst.api) {
        mapzen.whosonfirst.api.set_endpoint('https://' + api_host + api_path);
        mapzen.whosonfirst.api.set_key(api_key);
}
```

Change it so that it looks like this (remember the access token we generated for the API?):

```
if (mapzen.whosonfirst.api) {
        mapzen.whosonfirst.api.set_endpoint('http://192.168.0.47/api/rest/');
        mapzen.whosonfirst.api.set_token('87ffbaee33443e4b9b385606d6b9af28');
}
```

Note that we changed the `set_key` function call to `set_token`. This is an important difference.

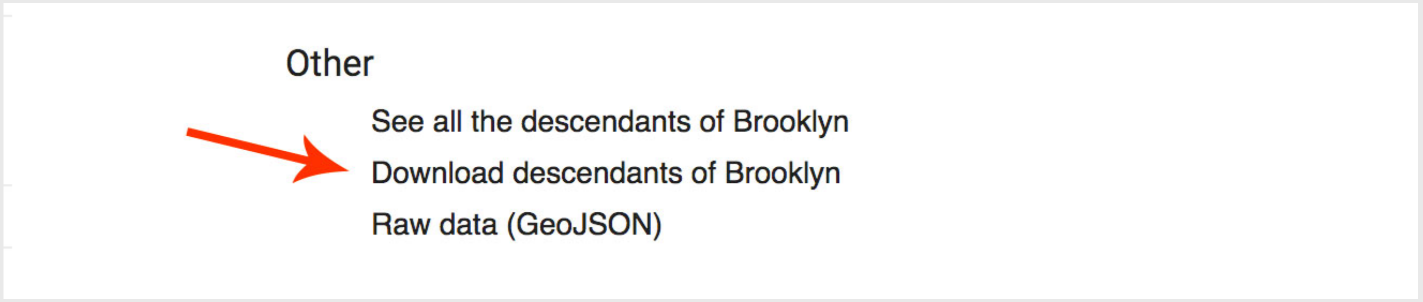
Okay, now we can try it all out!

First, restart the Spelunker for the config changes to take effect.

```
sudo /etc/init.d/whosonfirst-www-spelunker.sh restart
```

Next, load up the Spelunker page for Brooklyn (or Breuckelen). The URL should be something like `http://192.168.0.47/spelunker/id/421205765/`.

Now, scroll down until you get to the “Other” links, below the Hierarchy.

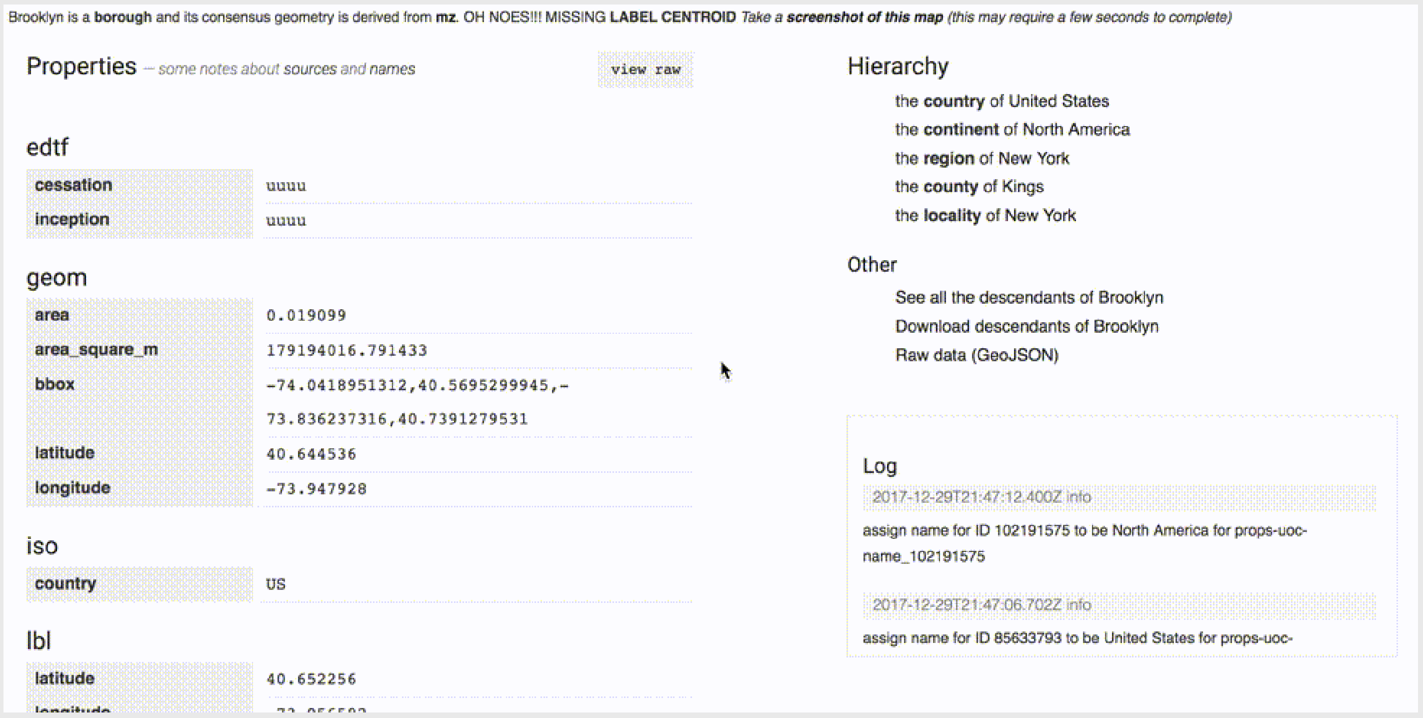


Other

See all the descendants of Brooklyn
Download descendants of Brooklyn
Raw data (GeoJSON)

You should see a new link for **Download descendants of Brooklyn**. Click on that.

If everything is working, you should now be able to bundle and download all the descendants records of Brooklyn, using the NUC's API and data. Let's try getting all the Brooklyn neighbourhoods.



Brooklyn is a **borough** and its consensus geometry is derived from **mz**. OH NOES!!! MISSING **LABEL CENTROID** *Take a **screenshot of this map** (this may require a few seconds to complete)*

Properties — *some notes about sources and names*

edtf

cessation	uuuu
inception	uuuu

geom

area	0.019099
area_square_m	179194016.791433
bbox	-74.0418951312,40.5695299945,-73.836237316,40.7391279531
latitude	40.644536
longitude	-73.947928

iso

country	US

lbl

latitude	40.652256

Hierarchy

the **country** of United States
the **continent** of North America
the **region** of New York
the **county** of Kings
the **locality** of New York

Other

See all the descendants of Brooklyn
Download descendants of Brooklyn
Raw data (GeoJSON)

Log

```
2017-12-29T21:47:12.400Z info
assign name for ID 102191575 to be North America for props-uoc-name_102191575

2017-12-29T21:47:06.702Z info
assign name for ID 85633793 to be United States for props-uoc-
```

That's all, folks

That was a whole lotta WOF in a Box, thanks for reading this far!

VAYAGIF.COM

(https://giphy.com/gifs/vtT53Z1YmKCxW)

· 29 December 2017 ·

Dan Phiffer

Dan is a data carpenter, which tbh is kind of a made up title. But he likes it all the same.

© 2017 Mapzen